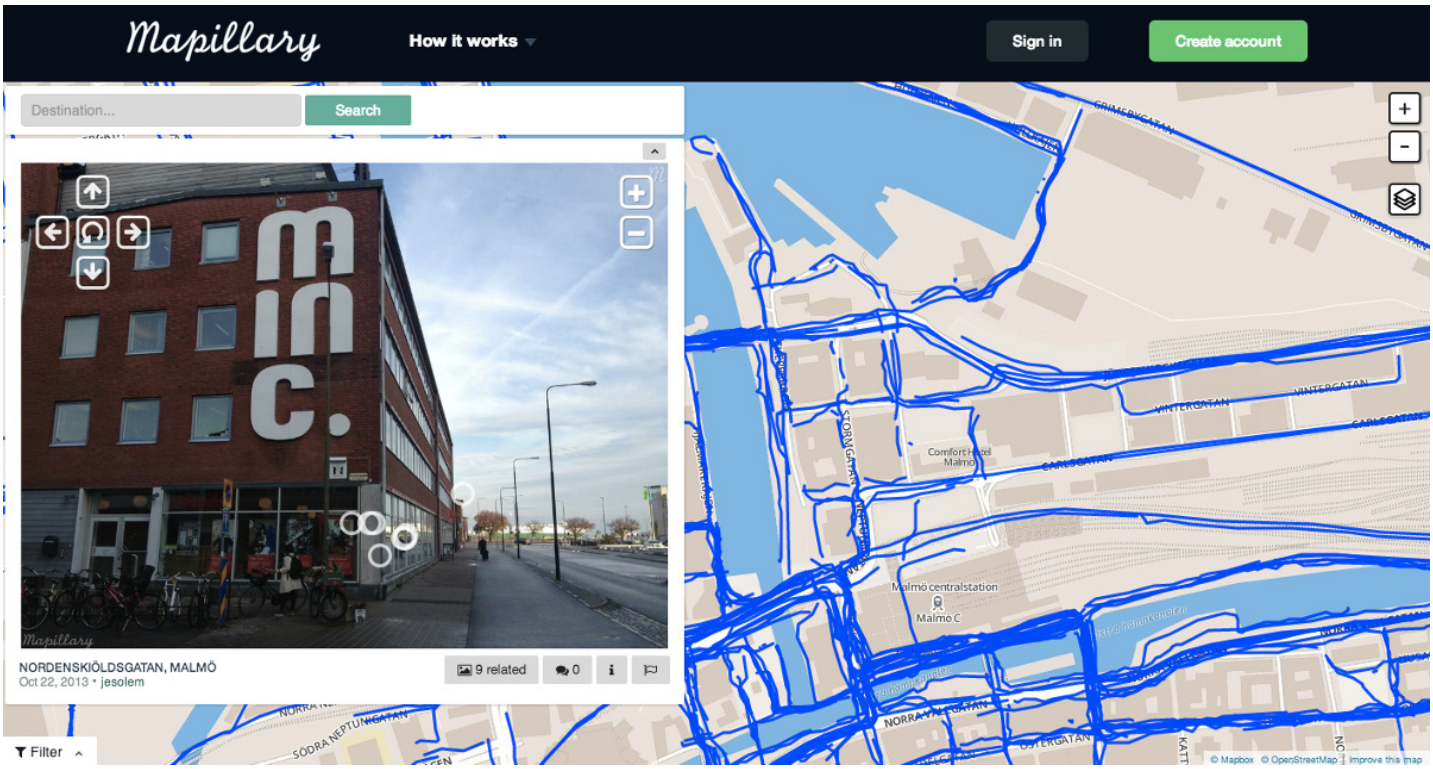
A Visit With Team Mapillary in Malmö

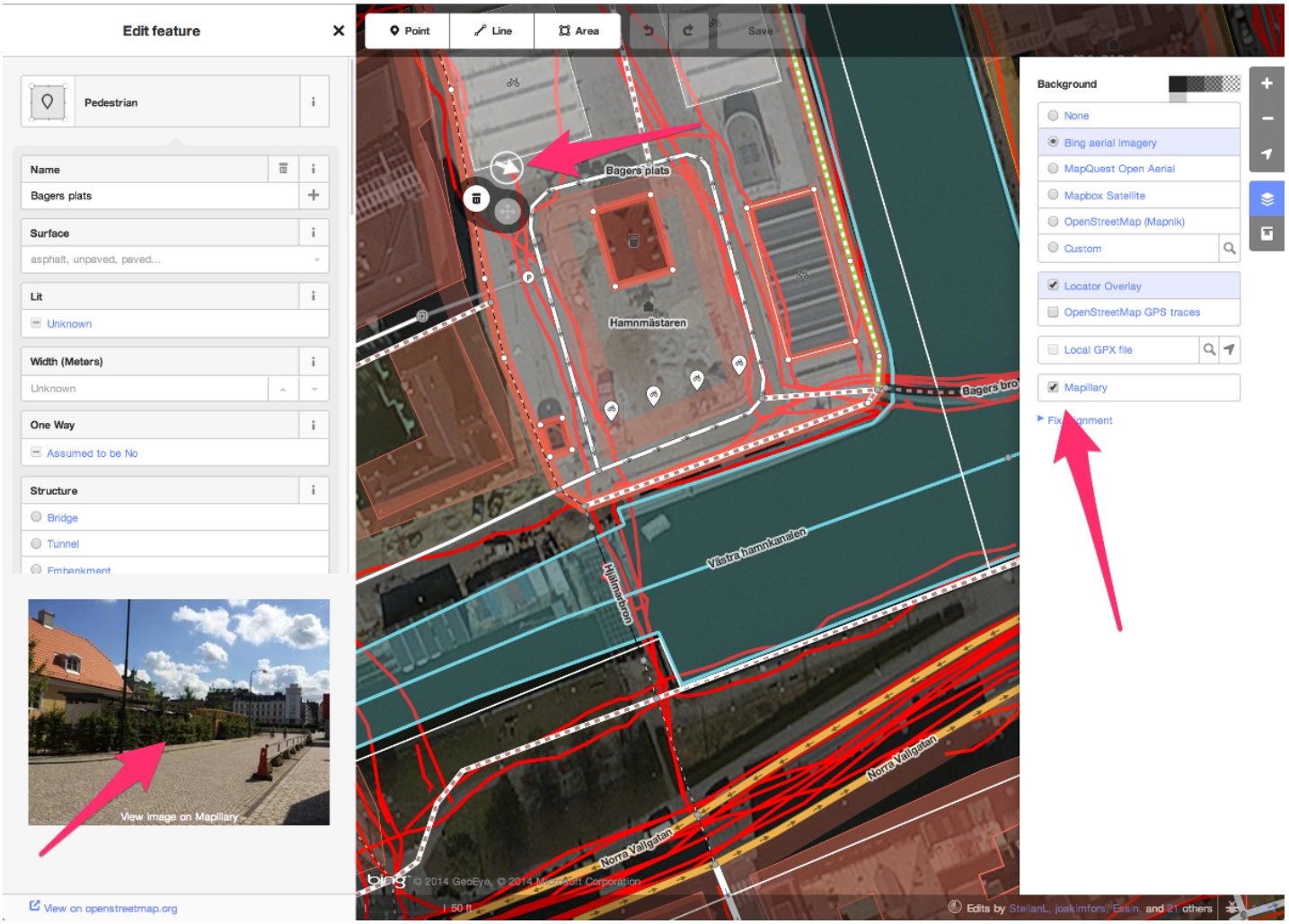
This past week I was lucky enough to be in Malmö, Sweden, where I stopped by the offices of **Mapillary (http://www.mapillary.com/)**, a company rethinking street-view imagery from, well, the street up. (Context: I was in Malmö as part of my other life where I make art, speaking on a panel at **The Conference (http://theconference.se/)**–which was also awesome, and I highly recommend the **videos here (http://videos.theconf.se)**.)



Mapillary is a resource for crowdsourced imagery of city streets, trails, and indoor environments from **all around the world (http://www.mapillary.com/map/places)**, made with the belief that people who live in a city can provide as useful, if not more useful, information about a place as a car passing through with a camera.

Crowdsourced imagery can offer more detailed, regularly updated imagery of a place, and humans can take photos of places and routes that cars might not access–narrow side streets and shortcuts, beaches and **hiking trails (http://mapillary.github.io/mapillary_examples/skaneleden-test1.html)**, or **train stations (http://mapillary.github.io/mapillary_examples/commuteskane-trip1.html)**.

This sort of information is not only super useful for non-driving **navigation applications (http://mapillary.github.io/mapillary_examples/routes-malmo_lund.html)**, but it's also valuable for verifying information about a place–which is why Mapillary recently created a **version of ID (http://blog.mapillary.com/news,/update/2014/08/19/id.html)**, the OpenStreetMap editor, that allows users to view available Mapillary imagery when editing an area.



All images added to Mapillary's public collection are released under a **Creative Commons Attribution-Sharealike (http://creativecommons.org/licenses/by-sa/4.0/)** lisence. You have the option to hide photos if you so choose, and they're currently working on providing a private repo service for things like organizations documenting infrastructure or construction projects over time. Images uploaded to Mapillary are processed to blur faces and lisence plates, just like that other Street View application you might know about.

I'm looking forward to seeing what people contribute to and build with Mapillary. They already have millions of images, and I'll probably be adding more of New York (you can get their app for **iOS (https://itunes.apple.com/us/app/mapillary/id757286802?mt=8&uo=4)** and **Android (https://play.google.com/store/apps/details?id=app.mapillary)**).

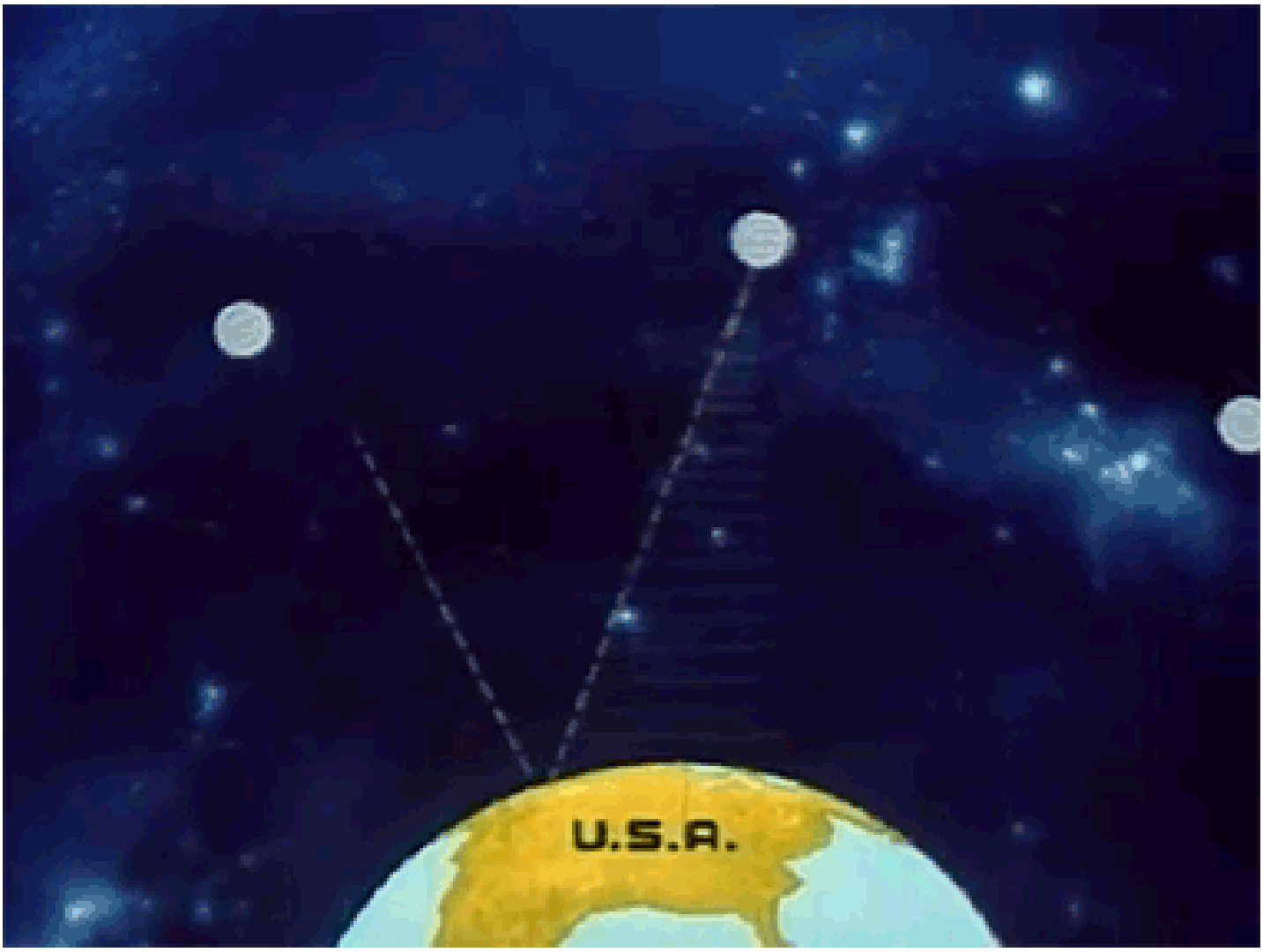
Take a look at some of their **project examples (http://mapillary.github.io/mapillary_examples/)** and their **API docs (http://www.mapillary.com/developer.html)** and see what you could do with Mapillary imagery. There's lots of potential–using computer vision to find and geolocate POIs, previewing the scenery of your proposed bus line made in **TransitMix (http://www.transitmix.net/)**, or showing changes to a neighborhood over time.

Thanks to Jan Erik, Peter, Yubin, and Peter for having me at the office–you are always welcome in New York!

· 22 August 2014 ·

Ingrid Burrington

GeoNYC Goes Out of Orbit September 16



We're looking forward to hosting this month's **GeoNYC meetup (http://www.meetup.com/geonyc/events/150268962/)** in our offices on Tuesday, September 16. September's meetup will focus on satellites–from giant ones to tiny ones to ones full of complex ethical implications. (OK, the ethics thing applies to big ones and small ones.)

Speakers include:

- **Lela Preshad (https://twitter.com/lelap)** of **NiJel (http://www.nijel.org/)**, who recently wrote a great piece for the **Tow Center for Digital Journalism (http://towcenter.org/sensors-and-journalism-the-people-within-the-pixels/)** on remote sensing that you should probably read

- **Kevin Bullock (https://twitter.com/kevin_bullock)**, a product manager at geospatial and satellite giant **DigitalGlobe (http://www.digitalglobe.com/)** and owner of a Minecraft t-shirt according to this video of his **State of the Map US 2014 talk (http://vimeo.com/91880883)**

- **Chris Holmes (https://twitter.com/opencholmes)**, an advisor at **Planet Labs (https://www.planet.com/)**, a company designing and building a network of small satellites and in possession of a really bold domain name.

Expect pizza, beer, and jetlagged post-**FOSS4G (https://2014.foss4g.org/)** recaps. Doors open at 6:30pm.

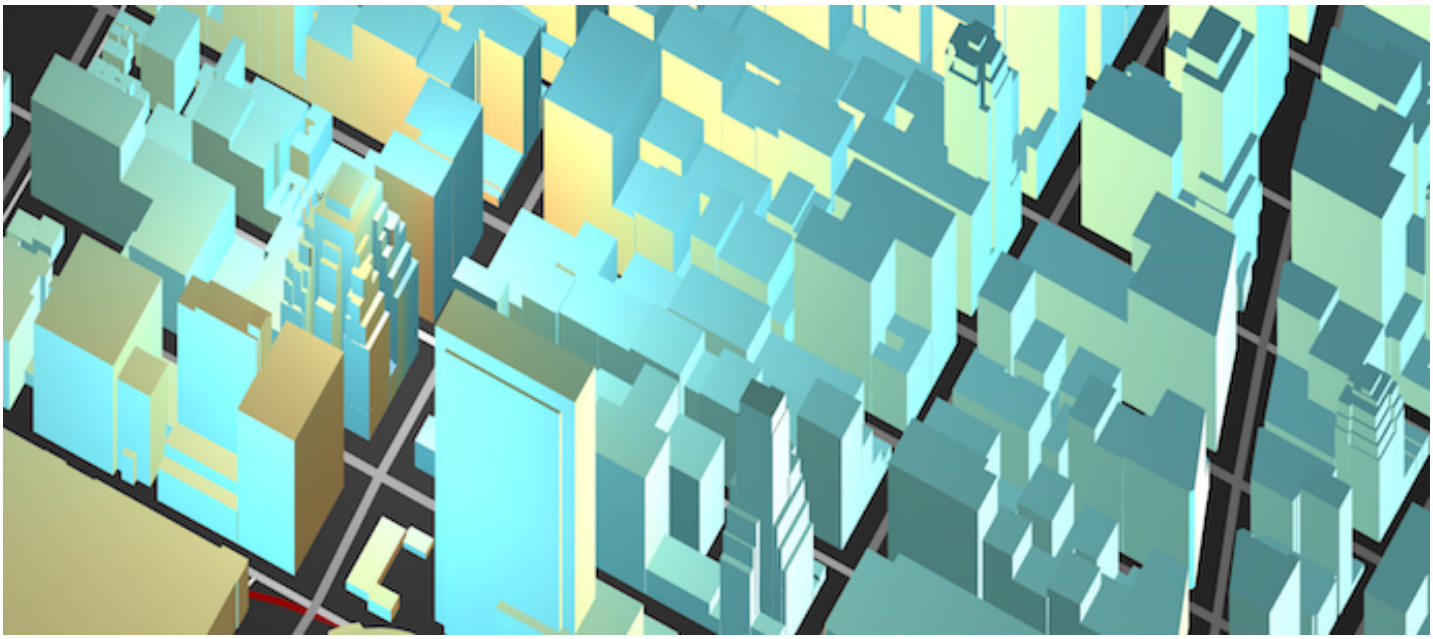
Any rumors that we will also be playing Dave Matthews Band's "Satellite" on a loop prior to the meetup are entirely false. We would never do that to anyone. (Obviously if you're making a satellite joke, Lou Reed's "**Satellite of Love (https://www.youtube.com/watch?v=MO5reyuzXis)**" all the way.)

· 25 August 2014 ·

Ingrid Burrington

Tangram, A Mapping Library

A New Look at Maps



(/projects/tangram/)

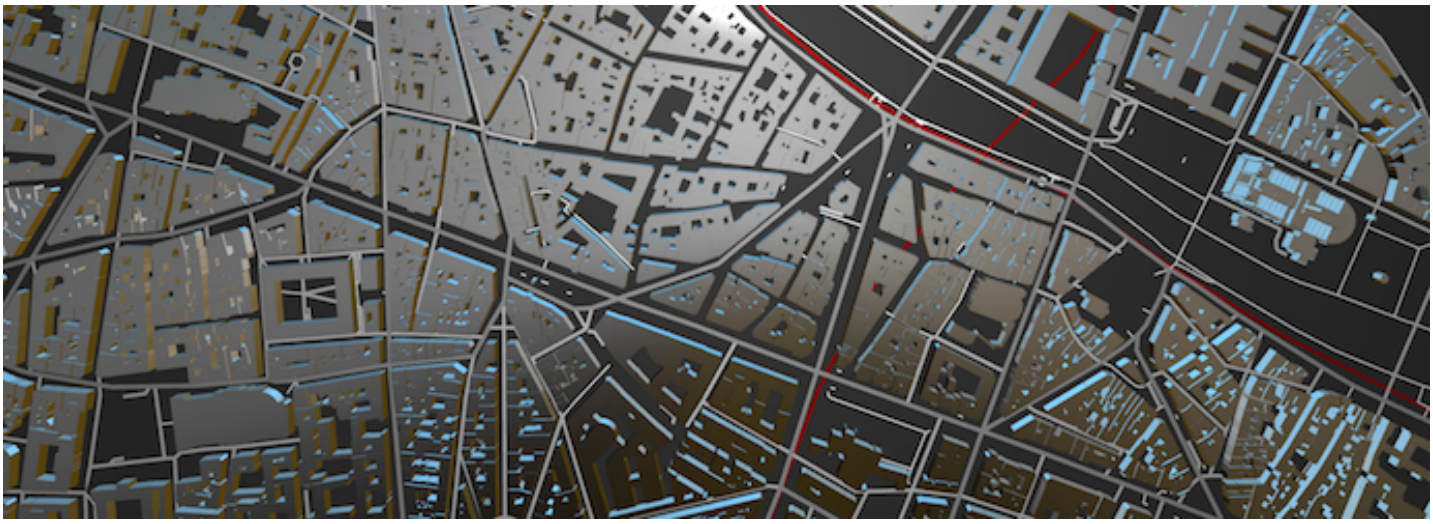
Traditionally, maps have been expensive and time-consuming to make, but that's changing fast. High-quality maps are easier to make every day thanks to OpenStreetMap and modern computer graphics capabilities, and we feel they can be pushed even further with ideas and techniques from 3D animation, video games, and other visual disciplines. To test this theory, the Mapzen graphics team has been working on a mapping library called Tangram.

Tangram (https://github.com/tangrams/tangram) is an open-source engine for real-time 2D and 3D maps. We're writing it around the **OpenGL (http://en.wikipedia.org/wiki/OpenGL)** family of graphics languages, starting with **WebGL (http://en.wikipedia.org/wiki/WebGL)** in browsers (soon expanding to **OpenGL ES (http://en.wikipedia.org/wiki/OpenGL_ES)** on mobile devices), and integrating it with popular mapping frameworks like **Leaflet (http://leafletjs.com/)**. Check out our **WebGL demo (/projects/tangram)** and read on for details.

Many map engines now use OpenGL to render familiar styles of maps, but we think OpenGL can do more. In particular, we've been inspired by the map customization demos of design studio **Stamen (http://stamen.com)**, such as their **Toner (http://maps.stamen.com/toner/)** and

Watercolor (http://maps.stamen.com/#watercolor) map styles. However, these maps are generated off-line, and served as static images; to change them, you must modify the style code, and then re-render the images, which can be a slow process.

We're using OpenGL to draw custom, programmatically-styled maps like these on the fly, which allows many more possibilities for customization, flexibility, and interactivity. We're big fans of live graphics-editing tools such as **Shadertoy (http://shadertoy.com/)**, and believe that many of the techniques demonstrated there can be put to practical use in the service of maps.



(http://tangrams.github.io/tangram/#mapzen,48.85305383712823,2.346117496490479,17,style=envmap)

Mapping libraries generally rely on a lot of assumed limitations to function. The ways colors are set, or lines are drawn, and even things as fundamental as map projection are sometimes hard-coded in, and there's no way to change any of it without tearing the whole thing down and rebuilding it.

We're designing Tangram to do all the things you'd expect from a mapping library, but constructed less like a black-box "solution" and more like a set of interlocking pieces, to be taken apart, rearranged, or replaced. Our goal is to make it possible to customize almost every step of the process – from cartography and traditional map styling to 3D geometry, lighting, and rendering. Our team has a wide range of backgrounds, from front-end and video game development to broadcast and feature film production, and we want to bring the same level of control we enjoy in other kinds of graphics tools to mapping.

We don't think ease of use and control are diametrically opposed. You can get started with Tangram in just a few lines of code, but beyond that, customization and styling control scales with your interest level and experience – you can specify the color of a data layer with a

stylesheet, get fancier with JavaScript styling functions, or go crazy and write your own shading effects in GLSL.

Ideally, Tangram will appeal to artists, designers, data visualizers, and coders of all stripes. We'll be continuing to demo features as we develop them, and we're especially interested to hear about features or capabilities you'd like to see in a client-side, 3D-capable, open-source mapping platform. We hope the possibilities lead to more exploration, more experimentation, and above all, more useful and beautiful maps.

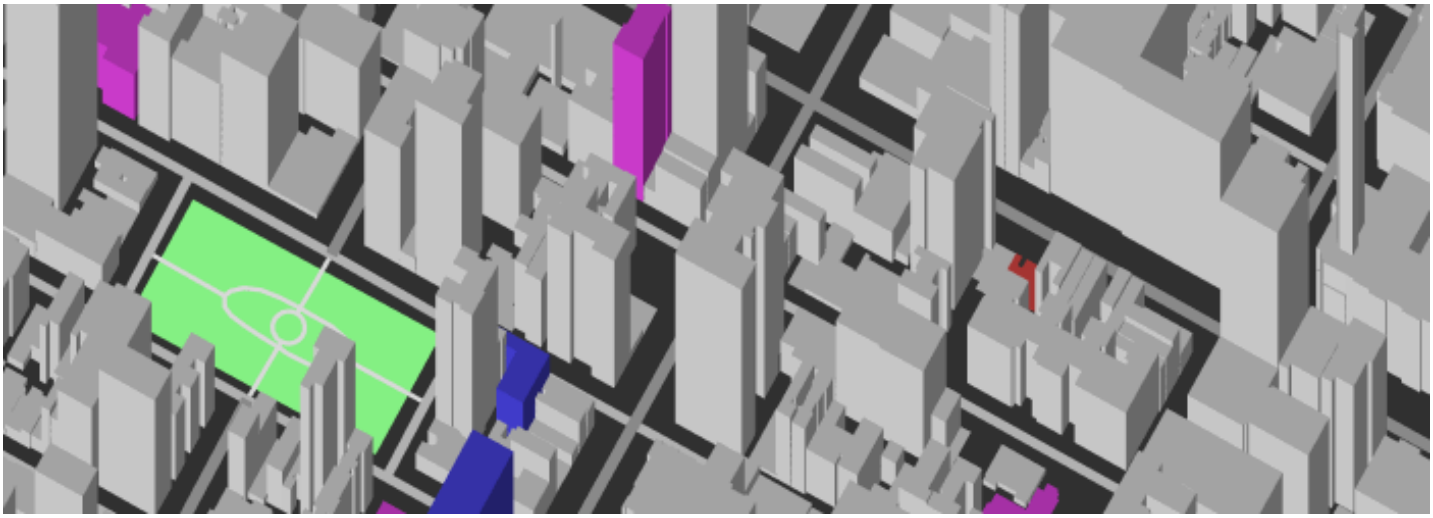
Follow along at our github repository **https://github.com/tangrams/tangram (https://github.com/tangrams/tangram)**.

On to the demos!

Gratuitous Map Gallery

These map styles are all written in GLSL; they can be referenced from a URL, or written inline in Tangram's "stylesheet".

Elevator

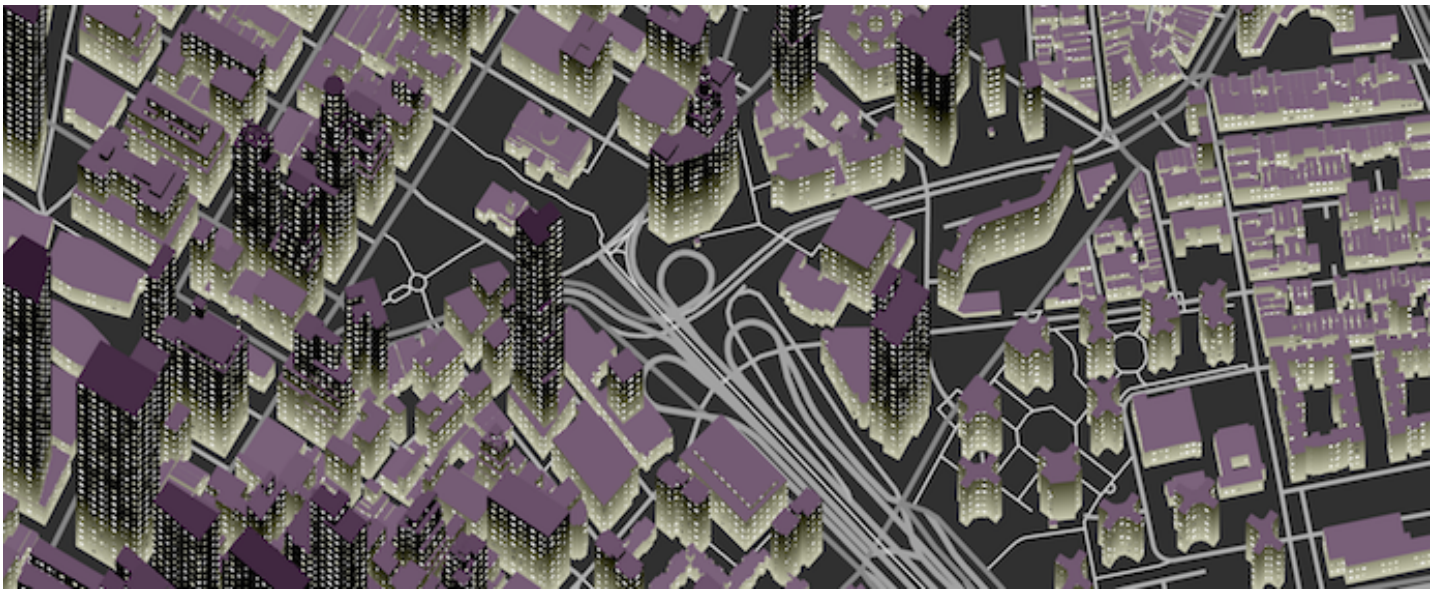


(http://tangrams.github.io/tangram/#mapzen,40.708279312457385,-74.00989294052125,16,style=elevator)

This mesmerizing effect modulates the height of all the buildings in the scene by a sine function. This can be accomplished with a single line in the stylesheet:

```
vertex: position.z *= sin(position.z + u_time) + 1.0;
```

Windows



(http://tangrams.github.io/tangram/#mapzen,40.71243508851969,-74.00450706481935,17,style=windows)

This tribute to SimCity references an external noise module.

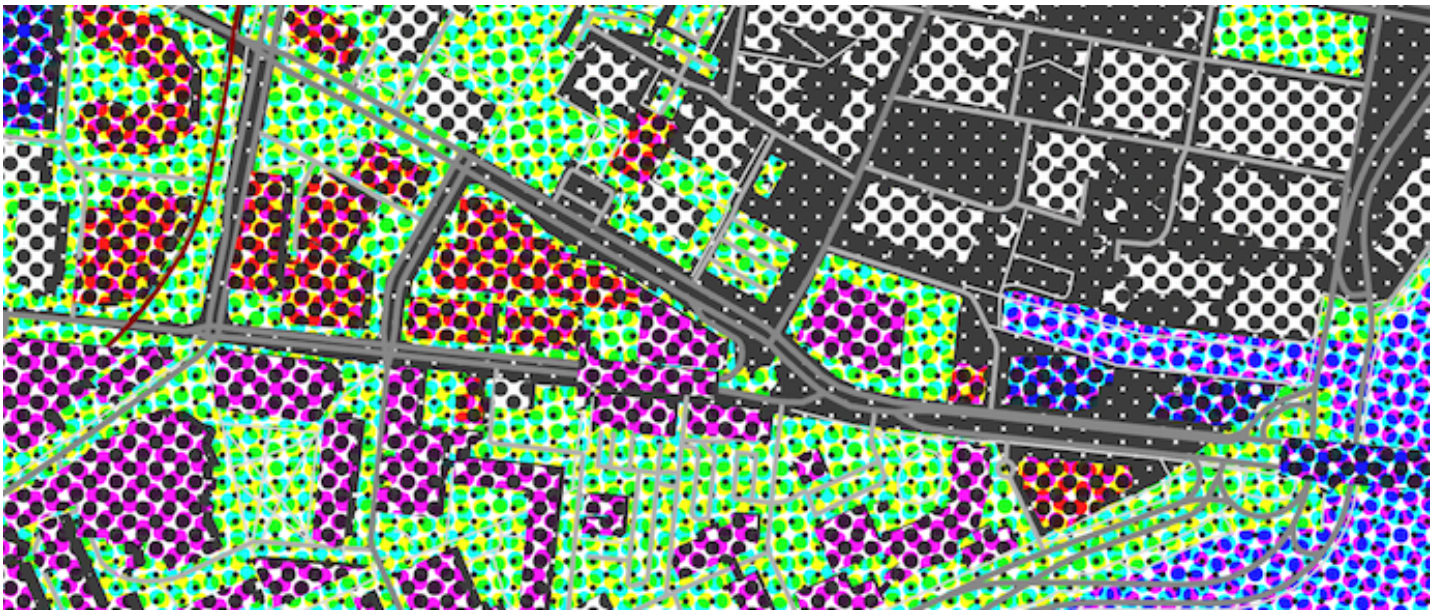
```
globals: |
    uniform float u_frequency;
    uniform vec3 u_windowColor;
    uniform vec3 u_buildingColor;
    uniform vec3 u_roofColor;

fragment: |
    vec3 vPos = worldPosition().xyz / u_frequency;

    // draw repeating bars on all axes, creating darker "window" voids between
    vec3 mask = mix(vec3(0.0), vec3(1.0), step(fract(mod(vPos, .9)), vec3(.4, .4, .6)));

    // treat brighter areas as building color
    if (mask.x + mask.y + mask.z > .5) {
        color = u_buildingColor;
        color -= vec3(vPos.z * .1); // height factor
    // treat darker areas as windows
    } else {
        // modulate the window color with a noise function
        float noiseColor = 2. * noise(worldPosition().xyz * 0.1 + (floor(u_time * 5.) / 1(
        color = u_windowColor * noiseColor;
    }
    // if the face points up, it's a roof
    if (v_normal.z > .6 || v_normal.z < -.6) {
        color = u_roofColor;
        color -= vec3(vPos.z * .01); // height factor
    }
```

Halftone



(http://tangrams.github.io/tangram/#mapzen,42.362293748983475,-71.08329176902772,17,style=halftone)

This shader emulates the CMYK-separation halftone printing process. Variables allow adjustments to the scale of the halftone screens, as well as to the dots themselves, which changes the apparent image exposure.

(This shader's a bit longer, so **here's a link to the code (https://gist.github.com/meetar/c67f8c0eff23f132990a)**.)

Take a look at our **github repository (https://github.com/tangrams/tangram)**, play with the **demo (http://tangrams.github.io/tangram/)**, and get in touch at **tangram@mapzen.com (mailto:tangram@mapzen.com)** if you have questions, ideas or screenshots of your own - we'd love to see what you come up with!

· 04 September 2014 ·

Peter Richardson

Peter vexes vertices, flusters fragments, and pesters pixels on Mapzen's graphics team.

Brett Camper

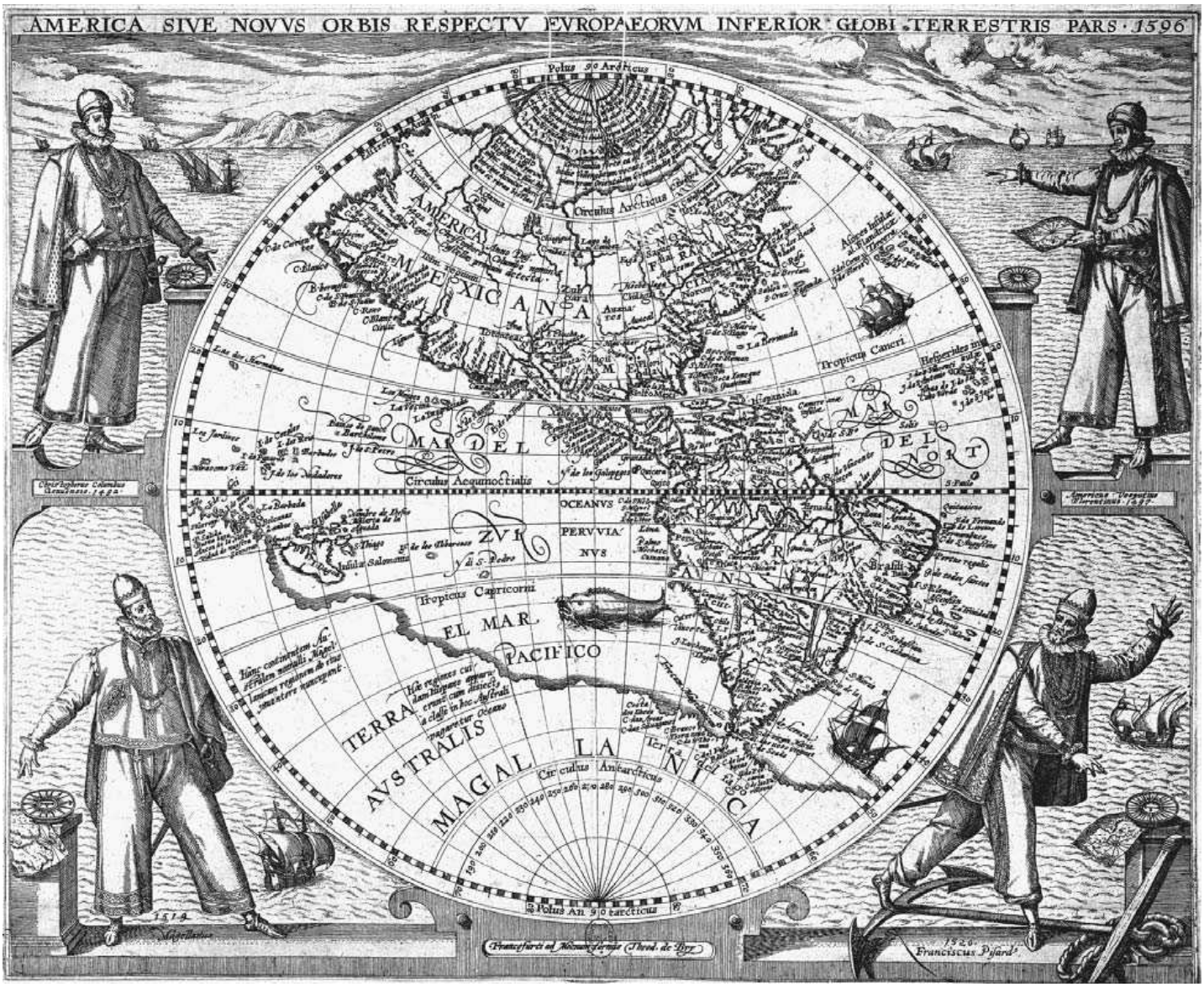
Starting Where We Are and Explaining How We Got There

I've been working with a **great team (/about)** to build Mapzen for the past year. When telling Mapzen's story, I usually start with my story of how I ended up in the open mapping world.

The first time I saw OpenStreetMap was in 2008, while working at a local news startup. We built our own map stack so journalists could improve the maps of their local communities. **Stamen Design (http://stamen.com)** worked on the cartography to match our brand, and I was introduced to the vibrant open mapping community at the time. OSM was good for display back then, but it was far from being a navigable, searchable map. However, anyone could see the rapid improvements that we and others were making.

Over the next six years, I continued to work with OSM in other roles at other companies, including facilitating MapQuest's move to using OSM data in 2010. At State of the Map in Girona, Spain, **we announced our support (http://readwrite.com/2010/07/08/mapquest_embraces_open_source)** of open mapping and worked to launch a number of tools, supporting the community in various ways. Working with OSM and its community made a huge impression on me and reinforced my belief that an open approach to mapping is the best approach to mapping. That belief informs all the work we're doing at Mapzen.

One thing I recall from back then: we looked at the quality of open data and wondered if it would ever be good enough for routing. Four years later, it's amazing how far we've come (**as Steve Coast notes (http://stevecoast.com/2014/05/19/why-openstreetmap-is-now-navigation-ready-for-people-like-you/)**). Today we look at the challenges of geocoding over open data, and I'm sure that we'll get there soon enough too.



Historically, mapping has been extremely difficult. This is true from the age of exploration up until the present, even as we see other technical areas becoming more accessible. A good map isn't just made by its software, but also its data. Even if you have access to the best search, rendering, and navigation software around, you still need a great global dataset to make any of it really useful. Plenty of companies work hard to offer some or all of these pieces, but usually with restrictions or at a price that makes it inaccessible to most developers and, ultimately, users.

At Mapzen, we start from the assumption that fully open software and data isn't just the best model, it should be the inevitable one. Ideally someday, companies and developers will work together on opening the inaccessible core of mapping technology. Instead of competing to have the best data or the fastest geocoder, they can compete somewhere more interesting, like their products.

We want to see what happens when we work only with open software and data to build professional-quality services and applications. Where are the weak spots with this approach? Although open mapping has made huge advances in recent years, there's still a lot of problems and a lot of work to do. Where can we help move things forward? Mapzen is an experiment for us, a test of our hypothesis that open maps are the future of maps. We're excited to contribute to a more open, more resilient mapping ecosystem.

This is where most blog posts on company websites usually say something about how "we're just getting started" or something. I'm not going to do that. I've been working towards this since 2008 so it doesn't really make sense. At Mapzen we often use the phrase "start where you are" to talk about our work. Right now, where we are is exactly where we want to be, and I'm looking forward to where we're headed.

· 08 September 2014 ·

Randy Meech

Billerica Memorial High School graduate. BA, MTS. Mapzen CEO 2013-present.

We're Donating Money to Code for America to Open Geodata. What Happened Next Will Shock You.

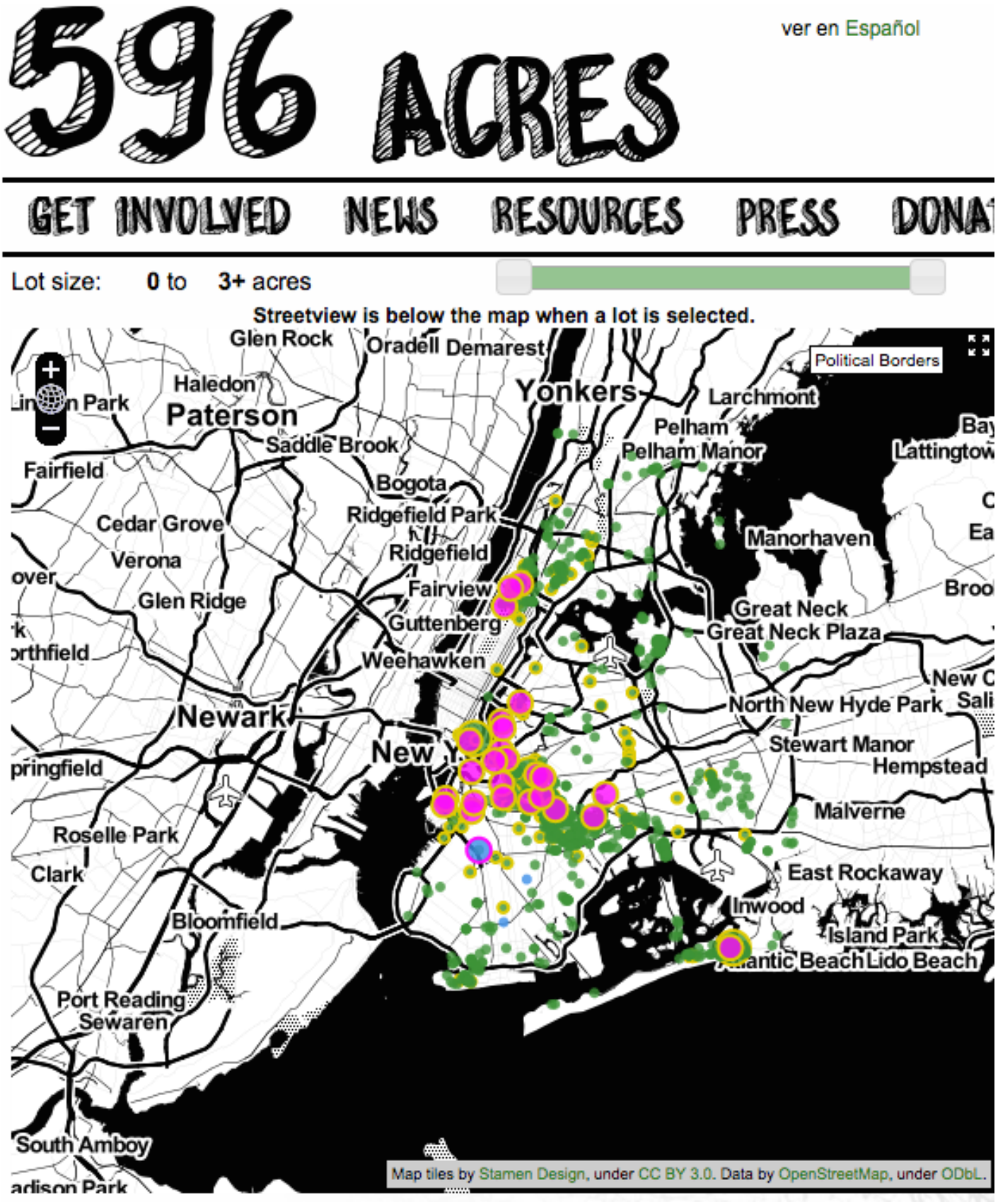
For the last 5 years, Code for America has connected governments and technologists so that both can better serve citizens. It's hard work. It's work that requires a lot of listening and learning, a lot of collaboration across organizations and agencies, and a *whole lot* of work with weird, hyper-particular government datasets. That's not always the kind of work that gets attention or accolades, but it is the kind of work that facilitates transformative change. And it's the kind of work we hope will happen through this **generous donation and partnership (http://www.codeforamerica.org/blog/2014/09/17/mapzen/)**.

Some of the key things this donation supports:

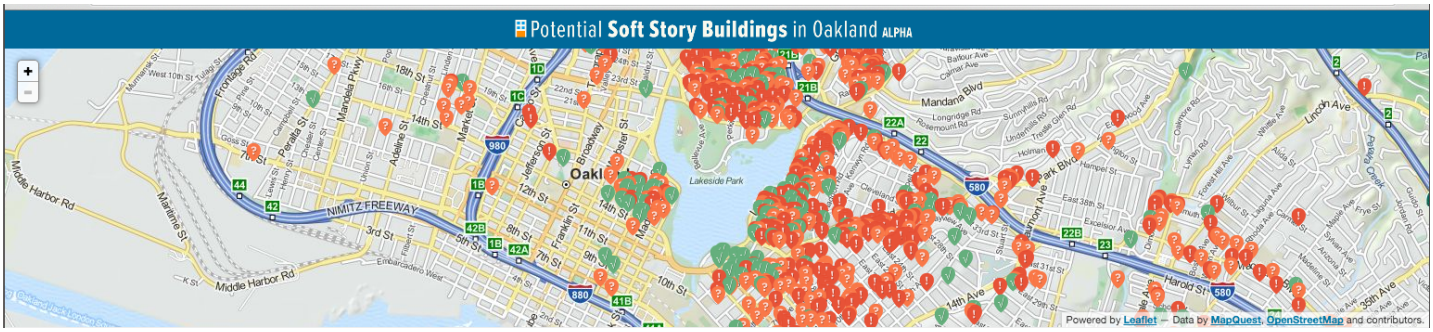
More open parcel data from cities.

Parcel maps often form the foundation for city mapping, and many city datasets have some geospatial component that relates back to that parcel map. When citizens and technologists have access to parcel data, they can do amazing things. Here are some examples of when community groups and organizations can use parcel data:

596 Acres (http://596acres.org) used NYC's parcel data, MapPLUTO, to help community groups gain access to vacant public land and build community gardens.



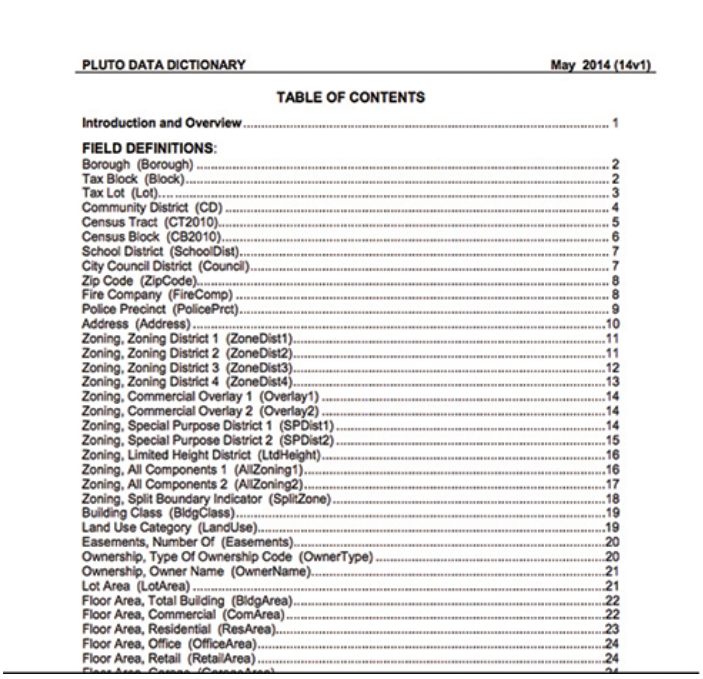
OpenOakland (http://openoakland.org), a Code for America Brigade, is using city parcel data to track **buildings vulnerable to earthquakes (http://softstory.openoakland.org)**.



This donation will contribute to Code for America's advocacy, education, and close work with cities on opening city parcel data. You can keep track of the inventory of open parcel data on the **US Cities Open Data Census (http://us-city.census.okfn.org/dataset/parcels)**, an initiative to track open datasets made in collaboration by Code for America, the Open Knowledge Foundation, and the Sunlight Foundation.

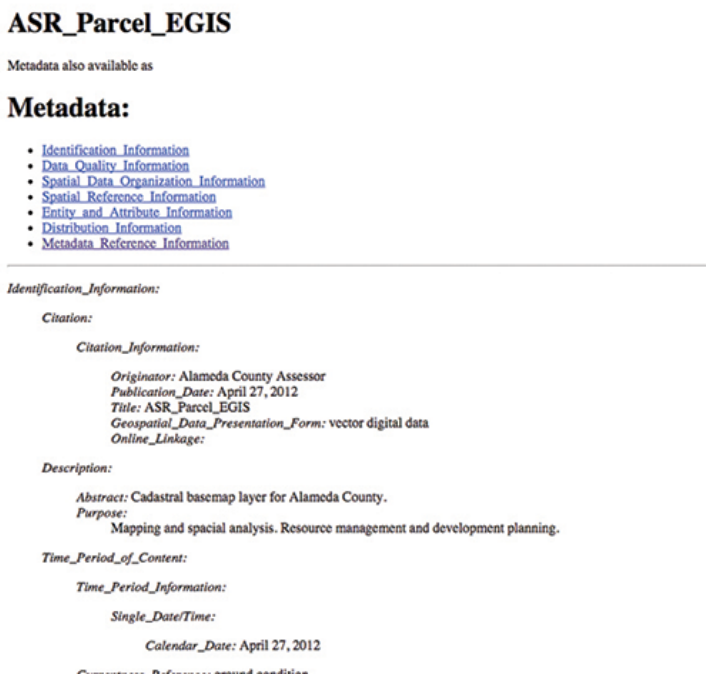
A common open format for parcel data.

Having more open parcel data is a huge deal. Making that data accessible and legible across contexts is potentially game-changing. Right now, every city releases parcel data in different file formats, using different terminology and formatting. Try reconciling the differences between **Alameda County (https://data.oaklandnet.com/Property/Alameda-County-Parcel-Boundaries/dnp4-5zvt)** and **New York (http://www.nyc.gov/html/dcp/pdf/bytes/pluto_datadictionary.pdf)** metadata:



PLUTO DATA DICTIONARY May 2014 (14v1)

TABLE OF CONTENTS



ASR_Parcel_EGIS

Metadata also available as

Metadata:

Identification_Information:

Citation:

Citation_Information:

Originator: Alameda County Assessor
Publication_Date: April 27, 2012
Title: ASR_Parcel_EGIS
Geospatial_Data_Presentation_Form: vector digital data
Online_Linkage:

Description:

Abstract: Cadastral basemap layer for Alameda County.
Purpose:
Mapping and spacial analysis. Resource management and development planning.

Time_Period_of_Content:

Time_Period_Information:

Single_Date/Time:

Calendar_Date: April 27, 2012

Such metadata. Much headers. Wow.

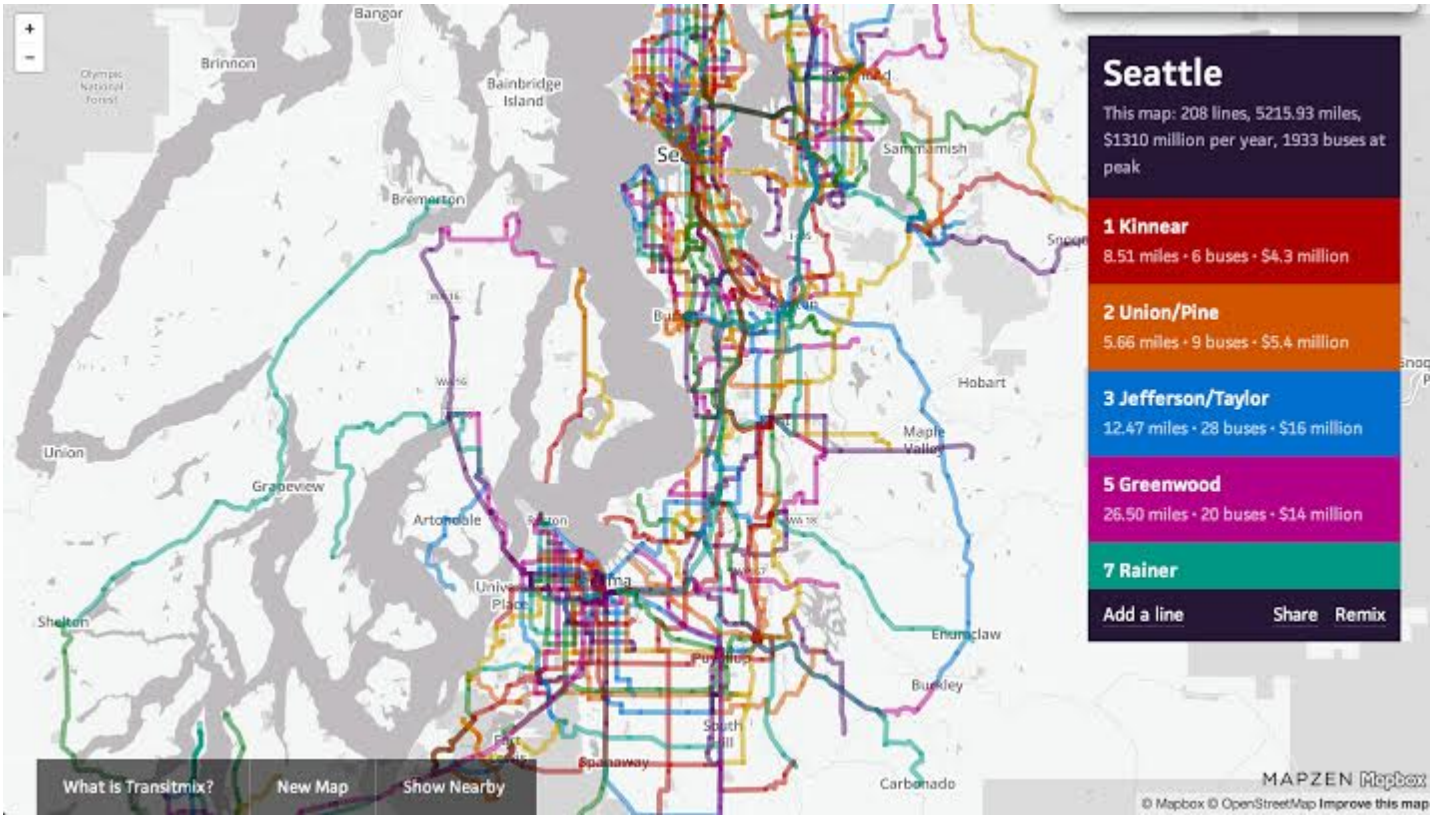
Furthering Code for America's ongoing work on **open data formats (http://www.codeforamerica.org/our-work/data-formats/)**, we're supporting the development of an open format for parcel data. The adoption of a common, open format—by governments and geospatial software platforms—could dramatically lower the barrier for entry to working with parcel data across contexts. Code for America is already working in this vein with the **Open Trail System Specification (http://www.codeforamerica.org/specifications/trails/)**, the first ever geodata standard to help public agencies open data about parks and trails and make it easier to digest and integrate into applications.

Brass tacks and infrastructure.

Geodata takes up a lot of space. (Ha, ha, ha. Space. Location data puns. But really though.) We're supporting the additional server costs and technical infrastructure needed to expand Code for America's work with spatial data. Maybe the servers will be in a data center like this one.

Focused work on mapping with and by Code for America Fellows.

Fellows come to Code for America from a range of backgrounds—sometimes with lot of open-source mapping knowledge, sometimes not. Since most city data is geodata, it's crucial to support Fellows with mapping tools and infrastructure, training and skillshares, and engagement with the larger open-source mapping community. We're supporting work by the CfA tech team to help Fellows produce more mature mapping applications. Our team will also engage in skillshares and hack sessions with Fellows, exploring open-source mapping projects.



We're already working with CfA fellows to help improve mapping applications—**TransitMix (http://www.transitmix.net/)**, a transit planning tool built by CfA fellows, uses our routing services, which we've customized for buses.

We're excited to work with Code for America on this initiative and look forward to kicking off this work with an **unconference discussion (https://cfasummit2014.pathable.com/meetings/228684)** on open parcel data at the **Code for America Summit (http://www.codeforamerica.org/summit/)** on September 24th and 25th. If you work with government geodata or have a government spatial data problem you're working on or want to see worked on, we'd love for you to be part of the conversation. Please reach out!

· 17 September 2014 ·

Ingrid Burrington

See you at Mapping Matters!

We're proud to be a sponsor of **596 Acres (http://596acres.org)**' gala **Mapping Matters (http://596acres.org/en/donate/gala/)** this **Thursday, October 2**, and hope you'll join us in celebrating and supporting their work.

596 Acres is among our favorite New York City open data projects. Using data from MapPLUTO (***before*** **(http://www.thenewyorkworld.com/2013/04/02/pluto-out-of-orbit/)** MapPLUTO was even open!) to identify publicly owned vacant lots, they help people find lots in their neighborhoods so they can turn those lots into community gardens and public spaces. Since they started keeping track of these lots in 2012, they've helped facilitate the creation of 28 community gardens in New York City, partnered with similar organizations in **Philadelphia (http://groundedinphilly.org/)** and **Los Angeles (http://laopenacres.org/)**, and worked on other amazing open data projects like the **Urban Reviewer (http://www.urbanreviewer.org/)**.

Also, there will be a roast pig! Tattoos! Maps! All of these things that you probably like. See you at **Galapagos Art Space (http://www.galapagosartspace.com/)** at 7pm this Thursday!

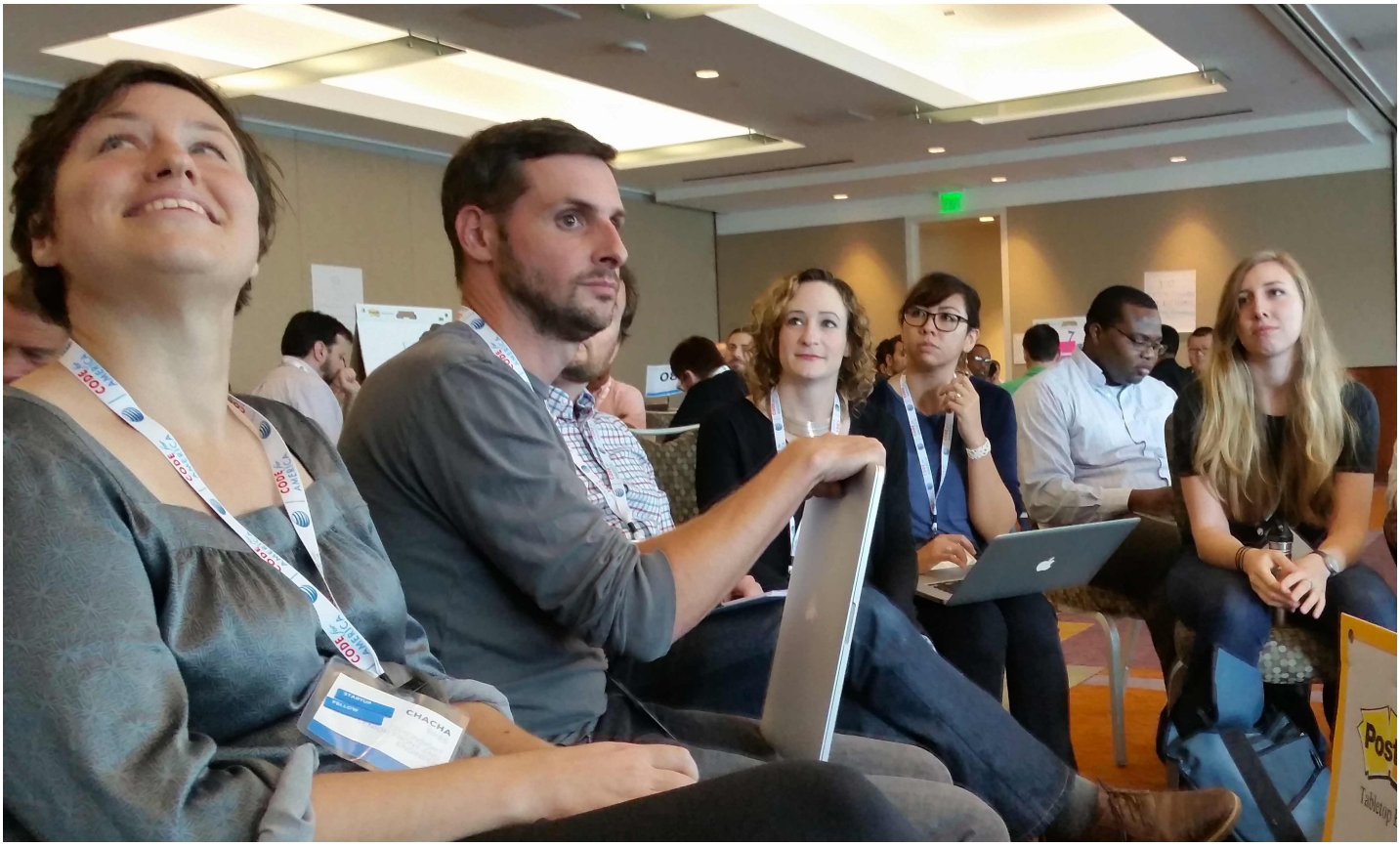
· 30 September 2014 ·

Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

Highlights from the Code for America Summit

The Mapzen team had a great time attending this year's **Code For America Summit (http://codeforamerica.org/summit/)**. It was an opportunity for us to learn more about work happening in civic tech, and meet members of the community, and look forward to the coming year **partnerning with Code for America (https://mapzen.com/blog/cfa-announcement)**. You can catch up on most of the talks by watching **videos (https://www.youtube.com/playlist?list=PL65XgbSILalWFStqV0z0N9pvftstJ8AAh)** from the summit, but here are a few highlights.



Our unconference session on **Using Maps for Civic Dialogue (https://cfasummit2014.pathable.com/meetings/228684)** attracted a great crowd, and there was a lively discussion on how the attendees from both tech and policy used maps in their own work. Many interesting topics were covered, but one of my key takeaways was a better understanding of the government agency perspective, which expressed portal fatigue and their disappointment over their data not being used after opening it.

One response to portals and their discontents might be doing what Mike "The guy in the above photo standing in front of a map" Migurski recommended in his session on **Defaulting to Open (https://www.youtube.com/watch?v=e2PMjB52q9c&index=44&list=PL65XgbSILalWFStqV0z0N9pvftstJ8AAh)**. Having the canonical data out in the open leads to a better ecosystem–there is no cost of "releasing" the data, as it's already out there. However, smaller municipalities will always have an uphill climb given that they don't have access to the same level of resources as bigger ones. The topic of smaller municipalities came up in a breakout session called **So You're Ready to Open Data: Now What? (https://cfasummit2014.pathable.com/meetings/228093)**. The session discussed different approaches to opening data and how to make sure that process is repeatable through automation and tools that a government agency already has.

There were many other awesome talks, sessions, and projects at the summit that you should definitely check out online. Ultimately, our team walked away from the summit hopeful. This community can solve a lot of problems. As was noted repeatedly at the summit, the problems that cities currently face are really, really big and really, really hard–most of which won't be solved with one technical fix. Luckily, the summit was full of people who understood that change doesn't come with just one awesome app or a data standard but with incremental, iterative, user-centered tools and services. We're looking forward to seeing what Code for America does this year with its new **partner cities (https://www.youtube.com/watch?**

v=iCcrEfblbcQ&list=PL65XgbSILalWFStqV0z0N9pvftstJ8AAh) and **focus areas (http://codeforamerica.org/our-work/focus-areas/)**, and we're looking forward to continuing the conversations that started at the summit.

· 02 October 2014 ·

Baldur Gudbjornsson

Former VP of Engineering, with special focus on devops, infrastructure, community and api management.

Join OSM!

Living in New York, I always like the voter guides they publish before elections, written in the most objective way possible and translated into the many languages people speak here. New York has been working on diversity and democracy for hundreds of years, and while it hasn't always been easy, "the more the better" is the prevaling attitude. Diversity is the city's biggest asset.

A lot of people who care about OpenStreetMap don't know about the elections, foundations, or the boards who run them. I didn't for a while. But I believe these organizations set the overall tone of the project and are crucial for engaging new and diverse members.

OpenStreetMap US

A bright spot here is **OpenStreetMap US (http://openstreetmap.us/)**. Over the past few years, the US foundation has grown into a large, diverse group that puts on successful conferences and engages new members admirably. **State of the Map US (http://stateofthemap.us/)** in DC this year was amazing, breaking all records for an OpenStreetMap conference, bringing in new people, and tapping into the excitement and possibility of the project.

Did you know that you can (and should!) run for the US board? All you have to do is **join here (http://openstreetmap.us/join/)** ($20 regular, $15 student) and then **sign up here (http://wiki.openstreetmap.org/wiki/Foundation/Local_Chapters/United_States/Elections/2014)**. The deadline to run is **today**!

Note that Mapzen's own **Alyssa Wright (https://twitter.com/alyssapwright)** is running, and we think you should too!

Not interested in running? That's fine, but we still encourage you to **join (http://openstreetmap.us/join/)** and vote!

OpenStreetMap Foundation (OSMF)

The **OpenStreetMap Foundation (http://wiki.osmfoundation.org/wiki/Main_Page)** is an international not-for-profit organization responsible for the OpenStreetMap project. It provides hosting support, fundraising, and organizes working groups necessary to keep the project

running. The OSMF also runs the international **State of the Map conference (http://stateofthemap.org/)**, which will be held this year in Buenos Aires from November 7-9. You should go!

While there are plenty of great people involved in the OSMF who do a lot of work to keep things going, in my opinion (and this is a blog post, not an election guide, right?) the organization isn't engaging newcomers or encouraging the growth of the project as well as it could be. There are, of course, many reasons for this. Ultimately, I believe that more people joining the OSMF will sort out a lot of the problems naturally; democracy has a way of doing that.

Most people don't know about the OSMF or how to join. But getting people who care about OSM to join is so important for the project and its success. So **sign up here (http://wiki.osmfoundation.org/wiki/Join)** for £15 per year. Run for the board, vote for the board, and keep engaged.

OpenStreetMap needs you signed up, engaged with, and voting for the international foundation (as well as your local group)! Join up and help out where you can. Like in New York City, if we all crowd in we can find a way to make it work better with diversity as our biggest asset.

· 02 October 2014 ·

Randy Meech

Billerica Memorial High School graduate. BA, MTS. Mapzen CEO 2013-present.

Getting Crafty With OSM Buildings

tangram (/tag/tangram)

OpenEverythingMap

OpenStreetMap (OSM) is the largest open-source map in the world. As you may surmise from its name, it notably maps streets. But the database holds more than just streets. Oh my, yes.

Every line in OSM is still officially a “way”, a nod to the OSM’s British origins, and the more commonly British usage of “way” to mean “road,” from the Old English “weg” which meant literally “road” (in the old English usage of the word “literally” to literally mean “literally”). In the original definition on the **OSM wiki (http://http://wiki.openstreetmap.org/)**, a “way” is “a street, footpath, railway line, etc.” But not all ways are ways anymore.

Over the 10-year history of OSM, a complex semantic ecosystem has emerged, repurposing the lines first used to draw streets to delineate everything else OSM volunteers wanted to draw. At this point, most of the “streets” of OpenStreetMap are no longer streets — they’re buildings.

Of all the mapped objects in OSM today, the most common is a building. Eyeballing the counts on **taginfo.openstreetmap.org (http://taginfo.openstreetmap.org)**, building data outweighs roadway data by a 3:2 ratio.

There are now nearly 130 million buildings in OSM. Many of these have been entered by hand, traced from satellite imagery, but recent large-scale imports of open-source municipal and national databases have added many more, including a million from the **NYC Building Footprints dataset (https://data.cityofnewyork.us/Housing-Development/Building-Footprints/tb92-6tj8)**, 18 million+ from the **Dutch national Registry of Addresses and Buildings (http://wiki.openstreetmap.org/wiki/BAG)**, and at least 35 million from the **French equivalent of the IRS (http://fr.wikipedia.org/wiki/Direction_g%C3%A9n%C3%A9rale_des_Imp%C3%B4ts)**.

In an attempt to learn more about these buildings, the ways that they are stored in OSM, and the ways they might be useful (and also to contribute to our maps puzzle table at the **Code for America Summit (http://www.codeforamerica.org/summit/)**), I decided to try to export a

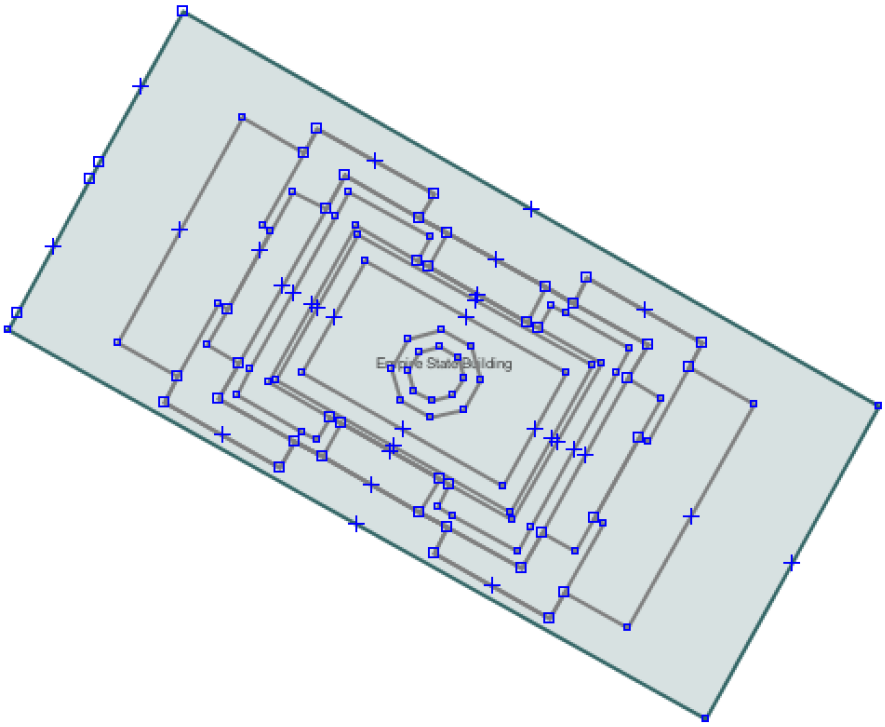
building and make a papercraft out of it. This turned into two very different problems, united by one common fact: OSM doesn't store buildings. Not really.

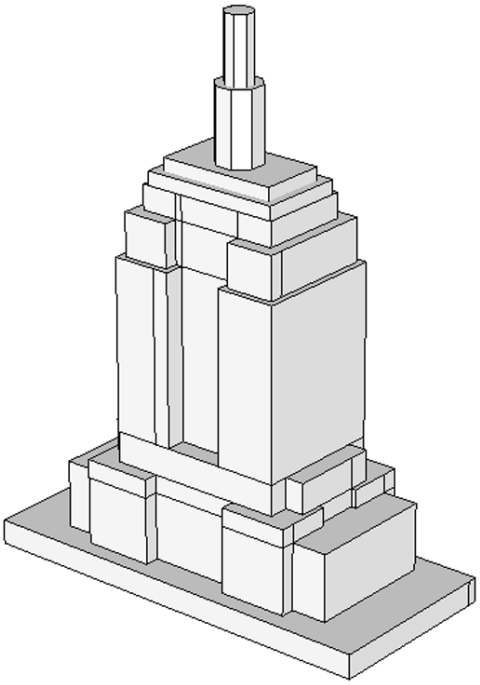
Flat Earth Society

So first, some mapsplaining. For a variety of reasons having to do with the nuclear strong force, the electromagnetic force, and gravity, maps are usually flat. Thus: OSM's data is generally assumed to lie on the plane of the ground. So, buildings in OSM are mostly just, like, outlines of buildings, drawn on the ground.

To describe things that aren't so flat, other kinds of OSM tags have come into use, such as the "height" tag, as well as the "min_height" tag which describes how far off the ground something starts, e.g. a specific floor of a building.

Mapzen's **Tangram WebGL map-drawing library (https://github.com/tangram-map/tangram)** uses this information to extrude these outlines upward, creating 3D objects which, at a distance, resemble buildings. I chose the Empire State Building because, Empire State Building. Also, it's easily recognizable and mostly made of flat surfaces and right angles. (I also squashed it to fun size, to maximize the fun.)





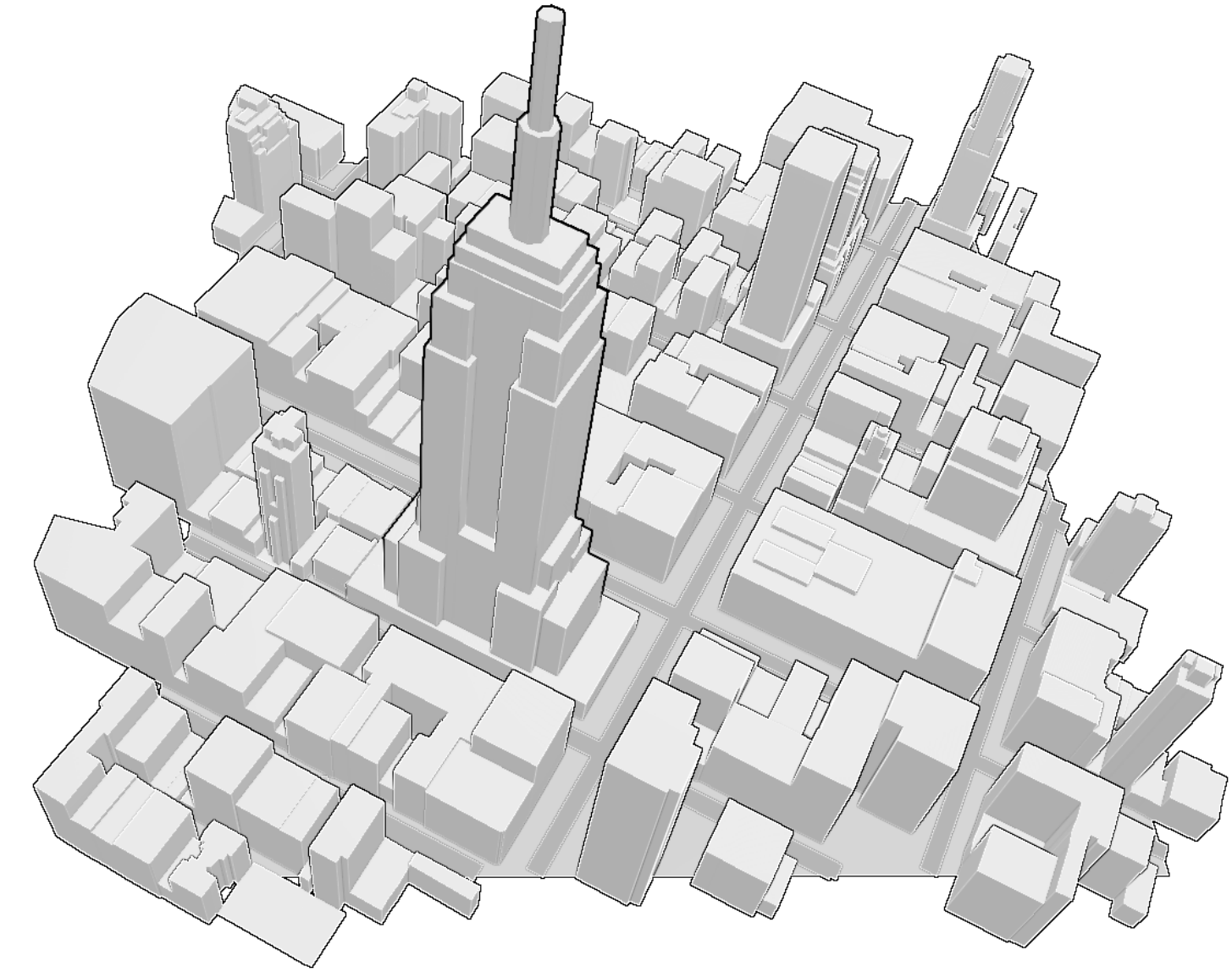
In most cases, the result satisfies our needs… but not always. I mean — have you ever really *looked* at a building? I mean *really*?

Well, if you ever have, you may have noticed that some buildings have more than one level of roof. OSM in its benevolent wisdom has provided another tag for these situations called “part”, to allow a building to be made from more than one “part”, as it were. Still other “relation” objects can be used to group these parts together into one conceptual building.

Many buildings in OSM are organized in this way, especially famous ones, which more frequently come under the scrutiny of eager mappers, and thus of OSM’s hoary cadre of data-purity enforcers. But in many other less-notorious cases, the buildings elude the gaze of the hoary cadre, and the data is messy, disjointed, or otherwise not up to spec.

Wild Tile

Complicating this situation is the fact that generally, this data is not accessed directly from OSM in its pure, crystalline form. OSM is a useful storehouse, but it isn’t designed as a high-performance production database. So, like many tools which use OSM data, Tangram makes use of a tile server, which decants bulk-exported OSM vector data into small, quantized “vector tiles” for improved access and transfer performance.



This means that to fetch a building, you must fetch the tile which holds the building — and very often, buildings span tiles. In Mapzen's tile server, a building "part" is copied into each tile it touches, and different parts of a building may be stored on different tiles. This is especially common for buildings with complex roofs or multiple towers, such as churches, castles, or fancy palace-type things.

Tangram, in its quest for expediency, ignores all of this semantic intricacy and blithely redraws any duplicated parts; adjacent pieces are drawn with their various tiles, buildings spring forth apparently solid and unharmed, and no one is the wiser.

This deception can cause headaches at export time if you're not expecting it, which is the first of the two sub-problems I mentioned earlier. But assume for now that something building-shaped has been extrapolated from OSM data. That's it — keep assuming for a few minutes while I describe 3D file formats.

Object Oriented

To humans, 3D objects are beautiful, complex things capable of eliciting unbearably strong feelings of attraction or disgust. But to a computer, a 3D object is just a bunch of numbers, grouped into sets of three. Call them “points.” When two points love each other very much, they make a line. And where three or more lines are gathered together, there a face is in the midst of them.

There are lots of 3D file formats, but at the core they all group numbers into sets of three to describe points. Some formats then group points into lines and lines into faces, but they’re still mostly just lists of numbers, so they’re relatively easy to write out and move around.

Not coincidentally, when Tangram extrapolates 3D geometry from the OSM data, the result includes a list of points. In the current Tangram scheme, these lists are stored in JavaScript objects, one object per layer per tile.

The process for converting these objects to 3D files (and indeed, from one 3D format to another) is mostly regexes and string shuffling. (You can check out the details in **this github repo: https://github.com/tangram-map/vbo-export/ (https://github.com/tangram-map/vbo-export/)**.) This process produces files which can be used in a variety of 3D apps, depending on the file type.

Once exported as an .obj and imported into **Sketchup (http://www.sketchup.com/)**, I was able to trim the tile of everything that didn’t look like an Empire State Building.

And thence we come in our journey — having left the Headwaters of OSM and the Polygonal Badlands — to the next great Realm of Problem: the Vale of Tears. Paper tears. You see, because I was tearing up. Paper. Let me explain.

Papercrafty

Papercraft is an ignoble art, the inverse of origami: rather than finding common cause with paper’s whims and humours through the subtle art of folds and bends, the paper is violently coerced to assume a shape contrary to its nature through brute force of cutting, and restrained from obeying the compulsions of its disposition with chains of strongest glue.

Naturally, as in all other areas of paper coercion, the Japanese lead the way. There’s one app in particular (unfortunately not open-source, which kind of sags my soapbox) called **Pepakura Designer (http://www.tamasoft.co.jp/pepakura-en/)**. It costs $35, and is written for Windows

(I used an emulator on my Mac), but there's really no other sane way to do it. (I gave that freeware Linux utility far more than a fair shot. It does not work good.)

Pepakura enjoys a large fanbase which has produced an **impressive body (http://www.mypapercraft.net/howl-moving-castle-papercraft/3617/)** of **tutorial (https://www.facebook.com/IronmanPepakuraTutorials)** and **instructional (http://www.instructables.com/id/What-is-Pepakura-and-how-to-start/)** content, so I will resist the temptation to add to it here, save for one detail: Pepakura, in order to discharge its duties, desires as its starting point a "solid," which is a concept in 3D with a number of uses.

A "solid" is a three-dimensional topological polyhedron, or polytope. Put another way, it's a three-dimensional orientable manifold with boundary. You will of course notice this implies that the Euler characteristic of the combinatorial boundary of such a polyhedron is 2. (For all you artsy types, this means the combinatorial manifold model of solidity guarantees the boundary of the solid separates space into exactly two components as a consequence of the Jordan-Brouwer theorem, thus eliminating sets with non-manifold neighborhoods.)

Put another way, a 3D solid is a single watertight piece, with no holes in the faces, intersections, subdivisions, or weird habits.

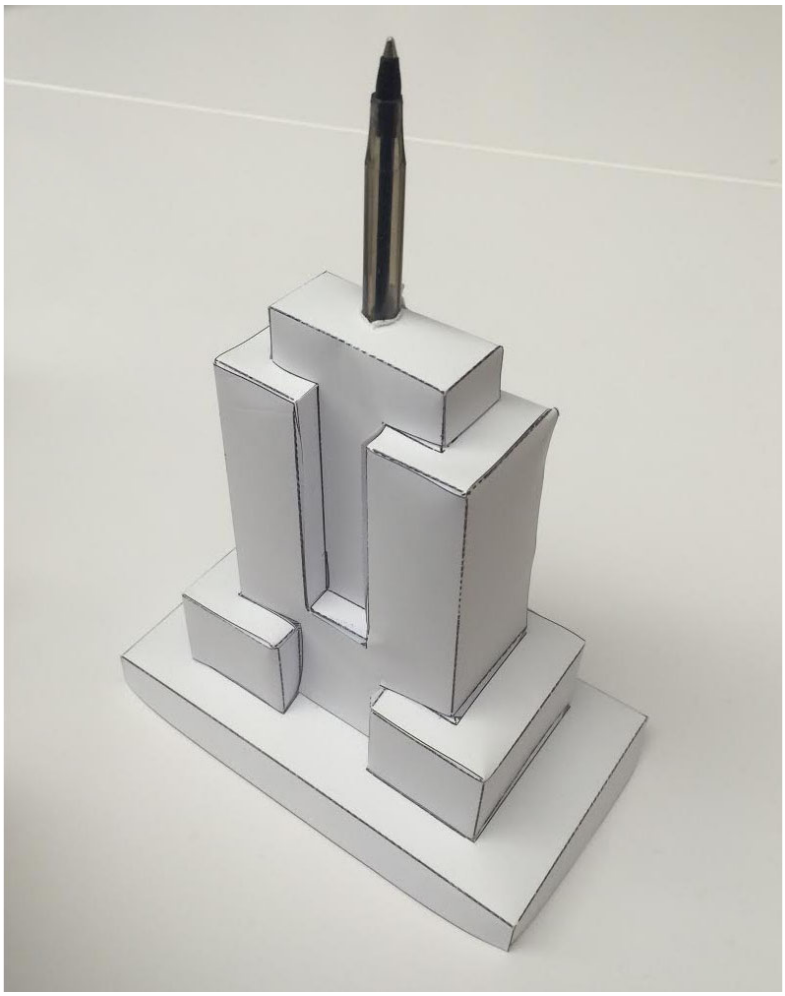
When Tangram builds a building, it extrudes the outline of each building or building part up to the appropriate height, then caps it with a roof face. Unless each part has a "min_height" tag, it is extruded from the ground up.

So our Tangram models are disqualified from solid status right away — they don't have a base because you'd never see it, they're full of extra faces, and they can have lots of intersections. They are far from watertight, and their Euler characteristics are hardly worth mentioning.

Our Tangram export function, so far, is perfectly happy to replicate this state of affairs, and the 3D file formats are (mostly) okay with it too. But this is not Pepakura-worthy.

Luckily, in a past life I was a professional 3D model pesterer, so I poked and tweaked and simplified the model until Pepakura accepted it as its own, and worked its mysterious magic, which is the true papercraft.

And it looks kind of neat! Very Empire State Building-like.



Conclusion

OpenStreetMap has lots and lots and lots of buildings. Most of them are either not so interesting or too complex for papercraft, and the vast majority don't even have a "height" tag. Still, exportable buildings could be interesting or useful when imported into 3D apps for custom rendering or printed in some way.

Regardless, our exporter doesn't currently immediately produce papercraftable models. And Pepakura isn't the only app which prefers solids to degenerate monster polys — many 3D printing services including **Shapeways (http://www.shapeways.com/)** like their models solid too. Extrusion printers such as the MakerBot may have an easier time with unmodified exported data, depending on the slicer software used to prepare the models, but it would be nice to make the data more useful in more situations.

So we plan to modify the exporter so that it produces less wacky geometry, and perhaps eventually add a step to combine overlapping shapes to create a single solid, which will make everyone happy at once.

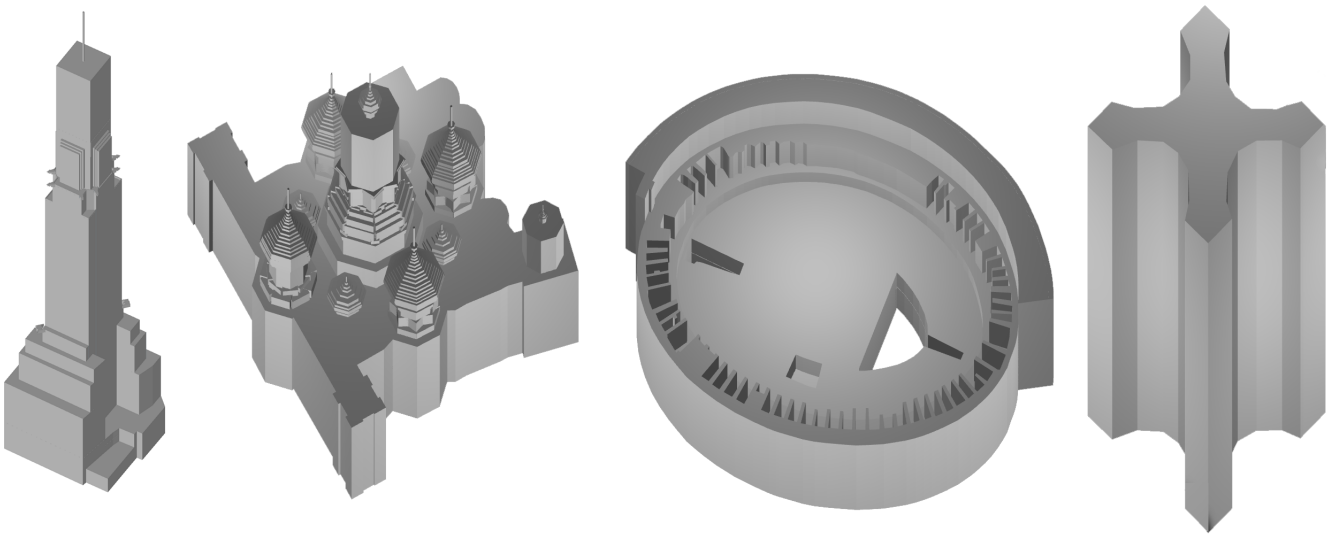
Parting Gifts

Here's a link to the exporter in its current stand-alone form: **https://github.com/tangram-map/vbo-export (https://github.com/tangram-map/vbo-export)**

Here's a link to the papercraft chibi-buildings I made, in three trendy color schemes based on a few of Tangram's procedural textures: **https://github.com/tangram-map/vbo-export/tree/master/ESB_layouts (https://github.com/tangram-map/vbo-export/tree/master/ESB_layouts)**

(If you attempt one of these, I recommend the also-not-open-source **Scotch® Adhesive Dot Roller (https://www.google.com/search?q=Scotch%C2%AE%20Adhesive%20Dot%20Roller)**.)

And to wrap up, here are some famous buildings, as interpreted by OpenStreetMap, to remind us all that there's still a lot of work to do.



Answers: Chrysler Building, St. Basil's Cathedral, Colosseum, Eiffel Tower.

· 03 October 2014 ·

Peter Richardson

Peter vexes vertices, flusters fragments, and pesters pixels on Mapzen's graphics team.

Apply to the GNOME FOSS Outreach Program with Humanitarian OpenStreetMap Team

The **GNOME project (http://gnome.org)** has been an amazing leader in expanding opportunities to underrepresented voices in open source. Through the **FOSS Outreach Program (http://gnome.org/opw/)**, GNOME facilitates paid 3-month internships for women (cis and trans) and genderqueer on open source projects. Past participants in the program include **The Open Technology Institue (http://oti.newamerica.net)**, the **Tor Project (http://torproject.org)**, and **Debian (http://www.debian.org/)**.

I'm really excited that Mapzen is sponsoring the addition of another amazing open source project to the program's roster–**Humanitarian OpenStreetMap Team (http://hot.openstreetmap.org/)**. HOT is an example of the OSM community at its finest–not only supporting on-the-ground work in crisis zones (such as their current work mapping regions affected by the **Ebola epidemic (http://hot.openstreetmap.org/projects/west_africa_ebola_epidemic)** in west Africa), but also supporting and nurturing new OSM communities in places like **Indonesia (http://hot.openstreetmap.org/projects/indonesia-0)** and **Senegal (http://hot.openstreetmap.org/projects/senegal_0)**. This internship opportunity is a really great way to get more involved with an **awesome team (http://hot.openstreetmap.org/our_board)** featuring some incredible women.

What You Need to Know to Apply

- Check out **HOT's wiki page (https://wiki.openstreetmap.org/wiki/Humanitarian_OSM_Team/OPM_Project_Ideas)**, which has more information about potential projects for the internship.
- Get involved! Reach out to the folks at HOT through their **mailing list (https://lists.openstreetmap.org/listinfo/hot)** or visiting their IRC (select the #hot channel **here (http://irc.openstreetmap.org/)**).
- GNOME has a fantastically detailed **guide to applying (https://wiki.gnome.org/OutreachProgramForWomen#Application_Process)** (complete with super-cute illustrations) that you should read!

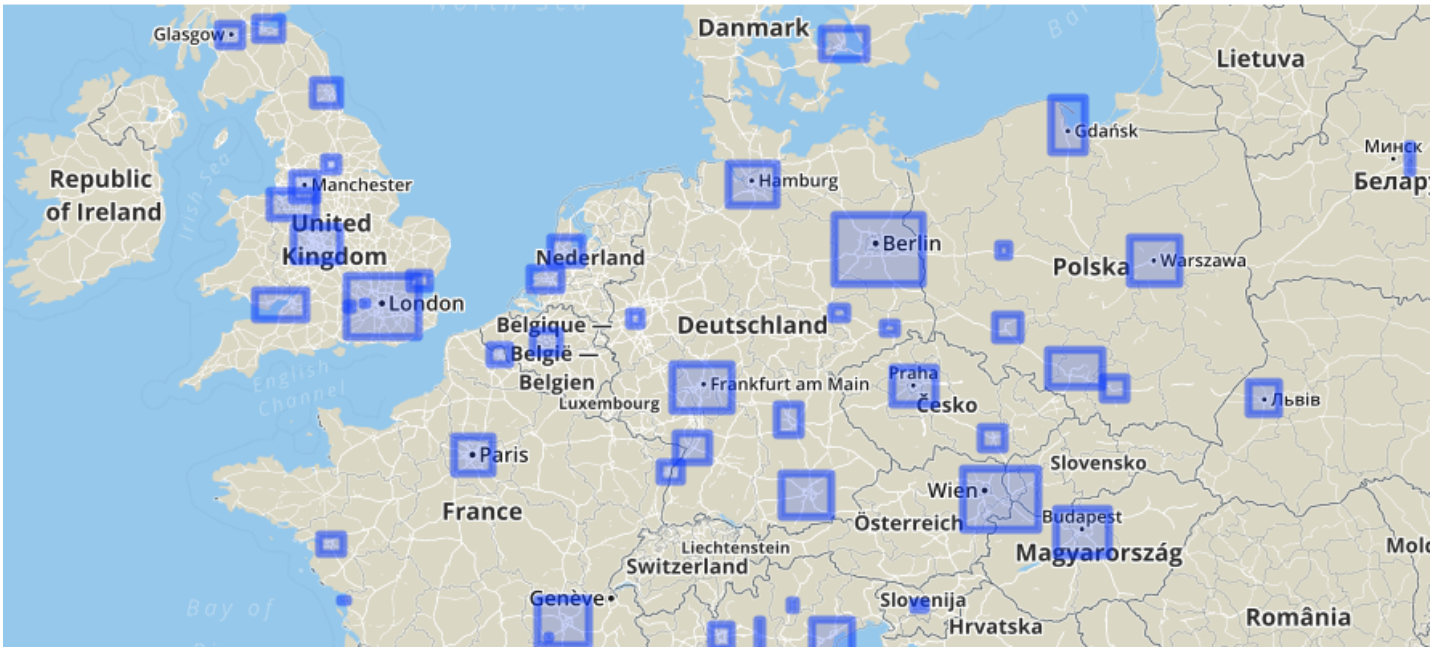
Having more diverse voices in the OpenStreetMap community has been a passion of mine for a long time, and OSM's own issues with diverse representation aren't unique within the **open source community (http://www.ashedryden.com/blog/the-ethics-of-unpaid-labor-and-the-oss-community)**. As I've described **previously (http://vimeopro.com/openstreetmapus/state-of-the-map-us-2013/video/68098504)**, the ramifications that lack of diversity has on OSM's data is kind of a big deal. GNOME's work is an awesome instance of the open source community genuinely putting its principles into practice, and I'm excited that Mapzen can contribute to this mission.

· 06 October 2014 ·

Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

So You Want To Use A Metro Extract



Mapzen's **Metro Extracts (https://mapzen.com/metro-extracts/)** download page accounts for a huge amount of traffic to our website—it's more popular than our actual home page. This is awesome, because they are a super-useful resource, and it's cool to see them being used by so many people.

For the most part, people visiting the Metro Extracts page know what they want and know their way around geodata. For the remaining two-thirds of our traffic that aren't going to the Metro Extracts page, you might not know exactly why or how to use them, which file format to download, or why there might be what look like inconsistencies in the downloaded data. This post is an overview of what metro extracts are, the file formats we offer, and what you might want to know to work with them.

Background: Metro Extract History

Mike "King Among Cartographers" Migurski initially created **OSM Metro Extracts (http://metro.teczno.com/)** in 2011, responding to a really basic problem: if someone wanted to use OSM data to make a map of, say, just the New York City metro area, she could either do a total planet download or download individual states from **Geofabrik (http://download.geofabrik.de/)** and cobble together a metro area.

Basically, a workflow that leads to making this facial expression:

So Migurski set up and, for a while, hosted Metro Extract downloads. And awesome people like **Nelson Minar (http://somebits.com/)** and **Smart Chicago (http://www.smartchicagocollaborative.org/)** and a ton of other people in the mapping community contributed to it and helped maintain it. With **Extractotron (https://github.com/migurski/Extractotron/)**, if that same user wanted to get OSM data of just the New York City metro area, she could do so really easily.

And in its time the Extracts came to reflect the size of the City, then the Province. And slowly but surely Migurski found himself building a map of the Empire which coincided point for point with Empire itself.

(Actually, it was just a kind of unwieldy project to maintain, but I had to hit my weekly Borges jokes quota.) Eventually Mapzen took on running and releasing Metro Extracts. We made a **chef recipe (https://github.com/mapzen/chef-metroextractor)** to do it (if those words just mean

food to you, **go here (https://docs.getchef.com/essentials_cookbook_recipes.html)**), which makes maintenance and updating the extracts easier.

Which brings us to now, and to our download page.

File formats

Let's say that I want to work on a mapping project and it only concerns New York City. Here's what I see when I look at the available NYC metro extracts:

The file formats as read from left to right basically go from the rawest form available for OSM data to the most structured, prepackaged format. (At Mapzen we make the "server farm to data table" joke a lot; in this case imagine we're moving from a bowl of coagulating soymilk to neat pre-packaged, pre-flavored tofu cubelets.)

OSM PBF and OSM XML

OSM is a special community. Likewise, OSM data is really special. So special, it gets its own file format that nobody else uses, **.osm**. These files can be compressed, either as XML **.bx2** or **.pbf**. There are grimier details of .pbf versus XML, for the purposes of this post let's just note that .pbf is smaller (more on .pbf **here (http://wiki.openstreetmap.org/wiki/ProtocolBufBinary)**).

Why would I want this format? Let's say I don't want to deal with messy admin polygons in my OSM data, and that I also want to filter for some specific tagged OSM data, like **amenity=police**. I could use some of the same command line tools that generate our Metro Extracts—like **Osmosis (http://wiki.openstreetmap.org/wiki/Osmosis)**, **osm2pgsql (https://github.com/openstreetmap/osm2pgsql)**, and **ogr2ogr (http://www.gdal.org/ogr2ogr.html)** to generate a GeoJSON with an OSM dataset custom to my needs. If you're real particular about what you need to extract from a metro extract, this is probably for you.

But if you want *everything* and if you don't really want to do more refining of the data yourself, maybe one of these shapefiles or GeoJSONs will serve your needs.

OSM2PGSQL and IMPOSM

If you're working with spatial data, you're most likely working with SQL data (listen, we can talk about hipster NoSQL stuff some other time, right now let's stick to file formats). **osm2pgsql** and **imposm** are tools for importing .osm data into PostGIS. Mapzen's chef recipe then generates shapefiles using the PostGIS command **pgsql2shp (http://www.bostongis.com/pgsql2shp_shp2pgsql_quickguide.bqg)** and GeoJSONs using ogr2ogr. osm2pgsql and imposm carve up .osm data in different ways that you can configure yourself; for now let's just talk about what Mapzen's configuration generates.

Our osm2pgsql export chops up OSM data into 3 datasets: **lines**, **points**, and **polygons**. Let's take a look at the point GeoJSON:

```
{
"type": "Feature",
"properties": {
    "osm_id": 368395980,
    "access": null,
    "aerialway": null,
    "aeroway": "helipad",
    "amenity": null,
    "area": null,
    "barrier": null,
    "bicycle": null,
    "brand": null,
    "bridge": null,
    "boundary": null,
    "building": null,
    "capital": null,
    "covered": null,
    "culvert": null,
    "cutting": null,
    "disused": null,
    "ele": "33",
    "embankment": null,
    "foot": null,
    "harbour": null,
    "highway": null,
    "historic": null,
    "horse": null,
    "junction": null,
    "landuse": null,
    "layer": null,
    "leisure": null,
    "lock": null,
    "man_made": null,
    "military": null,
    "motorcar": null,
    "name": "Unisys Heliport",
    "natural": null,
    "oneway": null,
    "operator": null,
    "poi": null,
    "population": null,
    "power": null,
    "place": null,
    "railway": null,
    "ref": null,
    "religion": null,
    "route": null,
    "service": null,
```

```
        "shop": null,
        "sport": null,
        "surface": null,
        "toll": null,
        "tourism": null,
        "tower:type": null,
        "tunnel": null,
        "water": null,
        "waterway": null,
        "wetland": null,
        "width": null,
        "wood": null,
        "z_order": null
    },
    "geometry": {
        "type": "Point",
        "coordinates": [
            -74.50099,
            40.3709408
        ]
    }
    }
```


So that's a lot of information to explain that this is a helipad. Basically every OSM tag that could be applied to a point, line, or polygon is stored as a feature property within that point, line, or polygon.

imposm exports are a little more granular—there are 18 separated datasets, most of which are kind of important OSM tags that intuitively make sense to separate out (administrative polygons, waterways, roads) and versions of the same dataset that have been "generalized"—i.e., simplified (if the filename has the suffix "gen" that's what it means).

So should I download imposm files or osm2pgsql? It depends what you want to do and whether you prefer a slightly more granular extract.

Some more persnickety projection information you might want to know

- *What projections do shapefiles and GeoJSONs use?*
  - imposm shapefiles: EPSG:4326 (EPSG:3857 for downloads prior to June 20, 2015)
  - osm2pgsql shapefiles: EPSG:4326
  - GeoJSONs (imposm and osm2pgsql): EPSG:4326
- *Why is the imposm projection different?* UPDATE: We're using **imposm3 (https://github.com/omniscale/imposm3)**, which previously supported only EPSG:3857.

Because the imposm3 export tool now has EPSG:4326 capabilities, we updated the Metro Extracts imposm shapefiles to use EPSG:4326, too. This means all extracted shapefiles and GeoJSONs use EPSG:4326.

Adding new cities

Adding a city to Metro Extracts is as simple as updating this **cities.json (https://github.com/mapzen/metroextractor-cities/blob/master/cities.json)** file and issuing a pull request. The cities are nested within regions (i.e. continents), and the **bounding box coordinates (http://wiki.openstreetmap.org/wiki/Bounding_box)** should be rounded to the third decimal place. If you don't know how to determine the bounding box for your metro area, try using the 'export' tool on **openstreetmap.org (http://openstreetmap.org)** (instructions **here (http://wiki.openstreetmap.org/wiki/Export)**).

Cool. If you landed on this page, you might be new to working with OSM data. Welcome to a weird, wonky world. It's got lots of helpful people and tools, and it probably needs you. Go forth and map.

Doing more with Metro Extracts

If you want to learn more about the Metro Extracts formats and what you can do with the data, follow this **tutorial (http://git.io/vmN0x)**. In the lesson, you will review the available file formats, load the Metro Extracts data into QGIS, perform attribute queries, and change the symbols used to draw the features.

You can find the tutorial at **http://git.io/vmN0x (http://git.io/vmN0x)**.

---

*This post, like all detailed introductory blog posts before it, is indebted to other detailed introductory blog posts. Fittingly, perhaps the most useful point of reference for this one is **by Mike Migurski (http://mike.teczno.com/notes/osm-and-postgres.html)**.*

Note: This post was updated on June 20, 2015 to include revised coordinate system information. It was also updated on July 27, 2015 to include links to a tutorial on using Metro Extracts.

· 10 October 2014 ·

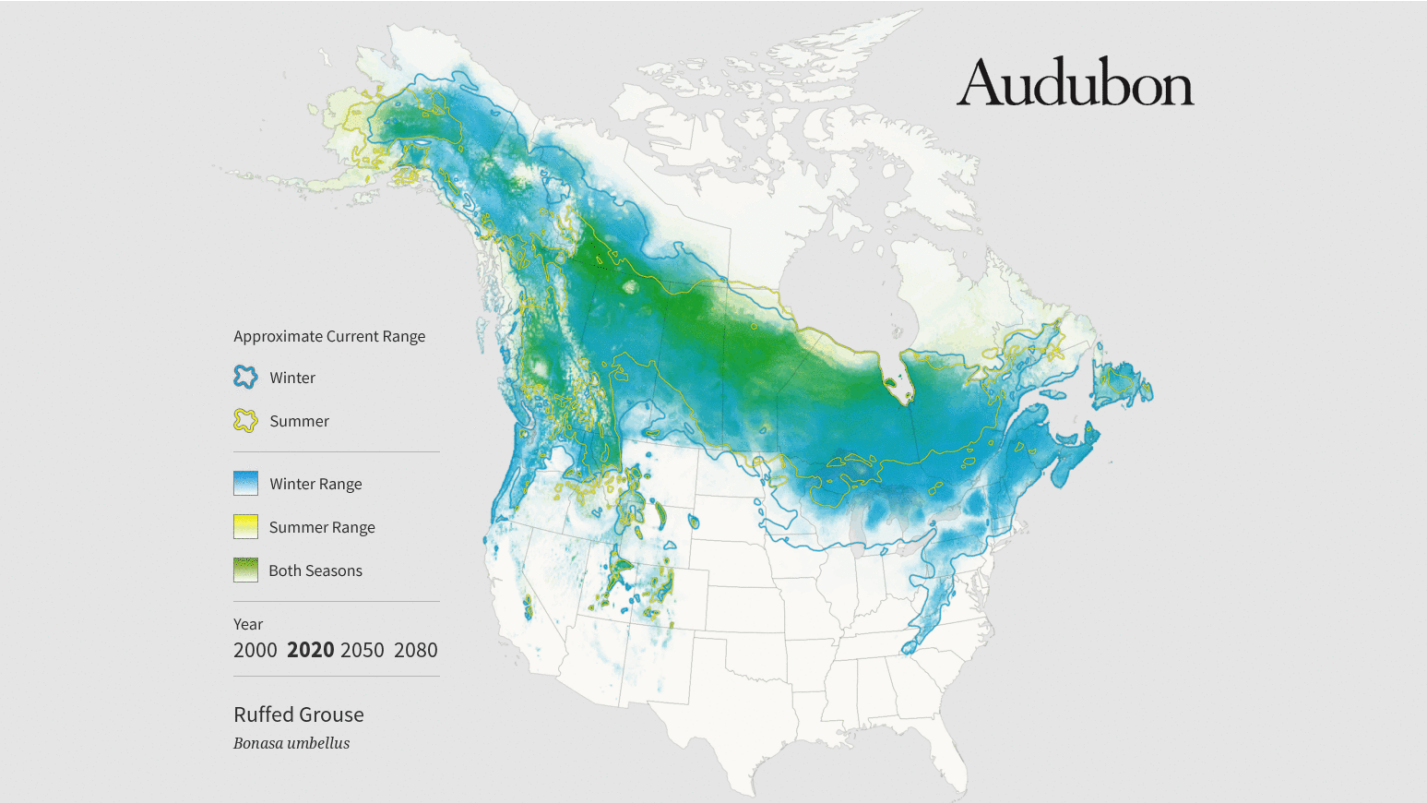
Ingrid Burrington

Maps are Full of Lies, and Other Thoughts from NACIS

I was in Pittsburgh last week at my first North American Cartographic Information Society **(NACIS) (http://nacis.org/)** meeting as part of the Mapzen team. The conference spoke to major themes at the intersection of maps and design and, for me, offered an opportunity to think about the way design informs how “true” we think maps are. The presentations and slides can be found **here (http://nacis2014.sched.org/)**.

Martin Elmer’s talk about ***Personality, Aesthetics, and the Human Touch (https://speakerdeck.com/maggiemacy/personality-aesthetics-and-the-human-touch)*** set the tone for the conference for me. Martin took everyone on a ride through history from hand drawn **historic maps (http://upload.wikimedia.org/wikipedia/commons/0/0d/Initialen.jpg)**, to **Edward Tufte (http://www.edwardtufte.com/tufte/graphics/minard_lg.gif)** and the emergence of the clean and “objective" aesthetics currently seen in most web maps. It made me think about whether online maps undermine the whimsical, emotionally resonant and subjective aspect of map making. Do people trust clean, objective maps more than hand drawn maps? (Should they?)

In his talk *Mapping with your Hands*, **Matt Dooley (http://mattdooleycartography.blogspot.com/)** shared his experiments with clay and gunpowder which reminded me of the highly artistic nature of maps. **Jake Coollidge (http://www.jakecoolidgecartography.com/index.html)**'s map of the Columbia River Watershed (also selected for Volume 2 of the **Atlas of Design (http://atlasofdesign.org/2014/10/07/volume-2-sneak-peak/)**) highlighted the core decisions cartographer's make while creating maps by hand like label placements, highlighting certain features and symbols. AJ Ashton brought the hand drawn sensibility to web maps with the **Pencil Map (https://www.mapbox.com/design/)** style made in Mapbox Studio. Alan McConchie talked about **Stamen (http://stamen.com/)**'s work with the Audubon Society using beautiful water color maps that showed the effect of climate change on bird ranges. The last two examples beautifully blended the world of hand drawn and tactile maps with digital web maps.



The tension between embracing subjectivity and demanding objectivity in maps is a theme that resonates with us at Mapzen, since **OpenStreetMap (http://www.openstreetmap.org/)** plays such a huge part in the work we do. If OSM is at all objective, it's because of the sum of thousands of subjective users' efforts. Every contributor has their own technique and perspective, but the map is governed by a community that (in theory) creates an objective standard for things like naming conventions and tagging.

Working with OSM data presents designers and cartographers with a unique challenge. How do designers honor the multitude of voices that make up OpenStreetMap while also making maps that are legible and functional? NACIS offered a lot of valuable, inspiring insights for taking on that challenge, and I'm looking forward to working on them.

· 17 October 2014 ·

Ekta Daryanani

Lead, Design + User Experience. Thinks: colors, people, community and city life. Does: mobile, maps, rock climbing and yoga.

OpenGhostMap: Making Haunted Tiles with OpenStreetMap data

(/ghost-map#ghosts,40.7298767593454,-73.91105890274049,16)

With the advent of Halloween, what better way to celebrate than with adding an element of spookiness to our tiles with a **ghost map (http://mapzen.com/ghost-map)**?

If you're not familiar with vector tiles, they're a way to break up map data into smaller pieces. Each tile represents a portion of the map, and requests are made to a back end service to fetch the data for each tile. Here's some more **details (http://mapzen.com/vector)** about our own vector tile service

We explored a few different implementation options for making our ghost map:

1. Find existing datasets of haunted houses, or other ways to correlate whether a particular feature could be considered haunted.
2. Randomly mark a feature as haunted using a deterministic approach from its data.
3. Based on OSM's existing tagset, develop a criteria to define whether a feature should be haunted or not.

Let's evaluate the different approaches.

Existing datasets

It turns out that there aren't many good existing datasets out there for haunted places that are easy to import in. Additionally, we'd like a balance of a healthy amount of places being haunted for Halloween, rather than having it be very minimal, or just in focused areas where data exists. Here are some notable links we did find:

- **A Wikipedia entry on haunted locations (http://en.wikipedia.org/wiki/List_of_reportedly_haunted_locations)**
- **This haunted house map (http://www.hauntedhouses.com/map.htm)**
- **hauntedplaces.org (http://www.hauntedplaces.org/)**
- **This awesome ghost map from Maptime HRVA (http://maptime.io/hrva/ghosts/hauntedHRVA.html)**

Random spookiness

The idea here was to take some part of the map feature's data, reduce it to a single value, and given a certain probability, use that to determine whether the feature should be considered haunted. It's important that this calculation be deterministic and not just totally random, otherwise a feature could toggle its hauntedness with every request. Although this could be a cool feature, ghosts aren't usually this fluid with their hauntings.

Haunted criteria

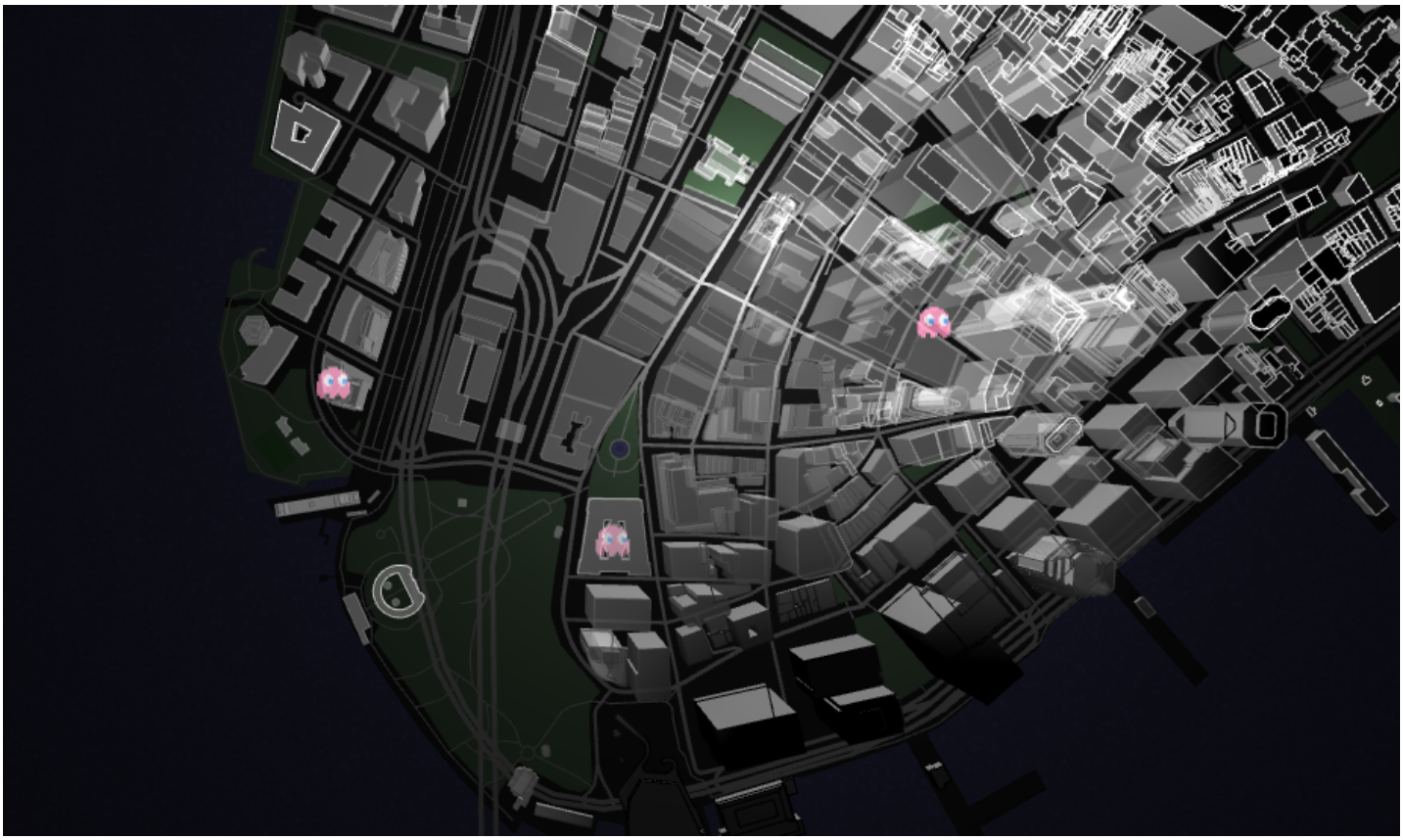
Here we define if a feature is haunted based on whether that feature contains certain tags in OpenStreetMap. We felt this was the most flexible, while also being faithful to the data source.

A feature could be a point, line or a polygon. Since we wrote a demo that renders points, we decided to explode the line and polygon geometries to produce a bunch of points. This is the reason you might see a group of ghosts hanging out together at a haunted cemetery. How do we determine if a feature is haunted? Unfortunately OpenStreetMap doesn't have a haunted tag

(although one was **proposed (http://wiki.openstreetmap.org/wiki/Proposed_features/Haunted_site#WOOOOO_THIS_WIKI_IS_STILL_HAUNTED)**). We chose a few different tags. Here's the sql snippet for our logic:

```
landuse = 'cemetery'
    OR amenity IN (
    'grave_yard',
    'crematorium',
    'library'
    )
    OR historic IN (
    'tomb',
    'castle',
    'archaeological_site',
    'ruins'
    )
    OR railway = 'abandoned'
    OR name IN (
    'Museo de las Momias de Guanajuato',
    'Weston State Hospital'
    )
```

Visualizing the data



(/ghost-map)

Our **ghostmap (http://tangrams.github.io/ghostmap/)** uses our **Tangram (https://github.com/tangrams/tangram)** library, which uses **WebGL (https://en.wikipedia.org/wiki/WebGL)** to draw vector map data real-time in browsers.

This particular demo uses a **sprite (https://en.wikipedia.org/wiki/Sprite_(computer_graphics))** drawn using our newly-added **texture coordinate (https://en.wikipedia.org/wiki/Texture_mapping)** functionality, as well as an **alpha (https://en.wikipedia.org/wiki/Alpha_compositing)** channel, modulated with a noise function referenced from an external file. (Note: for now, alpha is only available in Tangram's "alpha-dev" branch.)

The code for this demo is available **here (https://github.com/tangrams/ghostmap)**.

We're only going to have the ghost map up on our website for a limited spooky time, so if you want to do more with our spooky styles please check out the repo!

We hope you have fun exploring our spooky tiles and that they give you ideas for other uses of OpenStreetMap data. Happy Halloween!

· 29 October 2014 ·

Rob Marianski

Software engineer working on cutting tiles and infrastructure. Functional programming enthusiast.

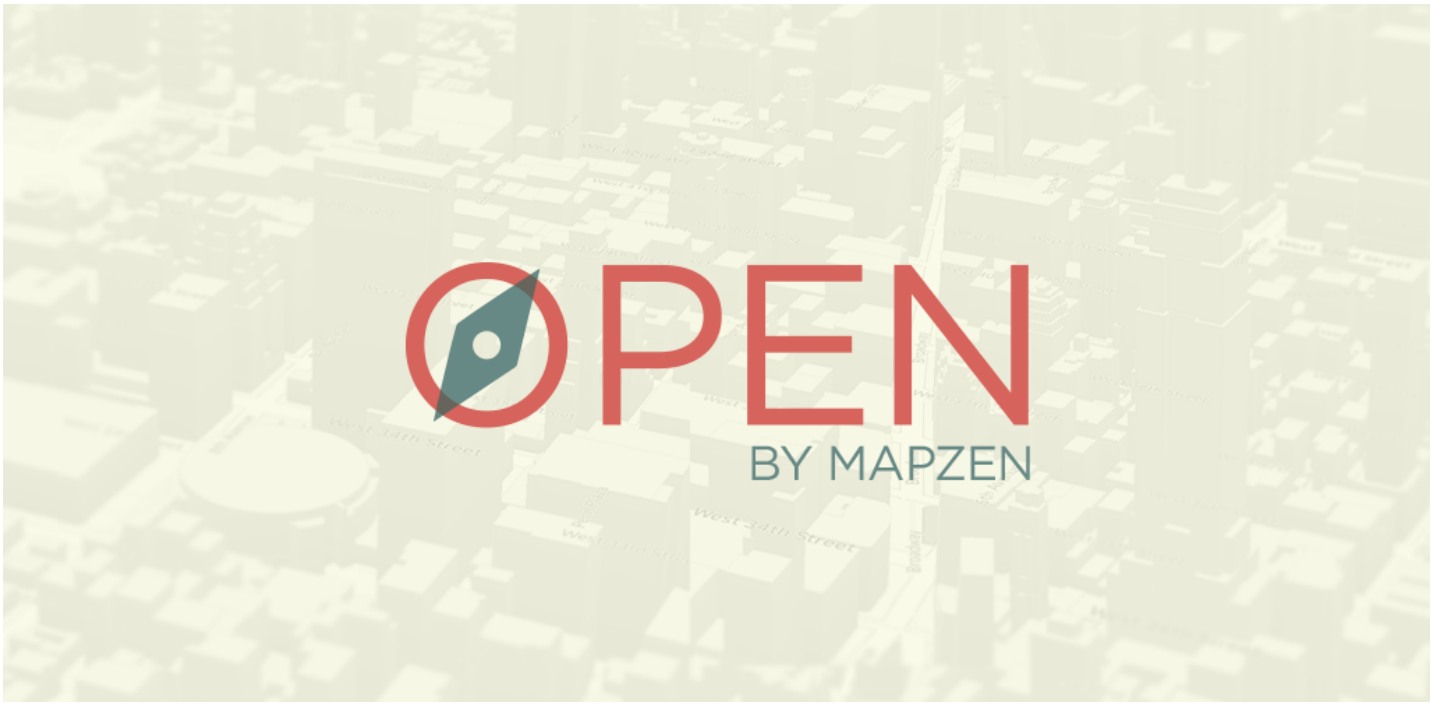
Harish Krishna

Peter Richardson

Peter vexes vertices, flusters fragments, and pesters pixels on Mapzen's graphics team.

Hey! We made an app.

mobile (/tag/mobile) android (/tag/android)



State of the App

Hi. So Mapzen's not an app company. We make open source tools and provide services for web mapping. But we believe that one of the best ways to improve open source tools is to put them in production—basically, eating our own dog food. Open source mapping has improved by leaps and bounds in the last few years, but in consumer mapping applications it's often still augmented with proprietary data or software. We decided to make a mapping app entirely from open source software and open data as a demonstration of the current state of open source mobile mapping and how it stacks up against its closed counterparts.

We named the app **Open (https://play.google.com/store/apps/details?id=com.mapzen.open)**, because that's what it's supposed to be. It's not *really* that open yet—currently it's available for use in North and South America, but we're adding support for more places over the coming weeks. It's also only available for Android because you know, open platform. If you don't feel like reading any amazing insights into how and why we built it, you can

stop reading and go download it in the Google play store **here (https://play.google.com/store/apps/details?id=com.mapzen.open)**, or if you prefer sideloading apps, the APK is **here (http://android.mapzen.com)**.

Open is built with OpenStreetMap at its foundation. We use OSM for the basemap, the road network for routing, and POIs for search. We're also using **OSM OAuth (http://wiki.openstreetmap.org/wiki/OAuth)** as a requirement for using the app—you need to have or make an OpenStreetMap account. We made that a requirement because while the app is in navigation mode it can optionally upload a **GPS trace (http://wiki.openstreetmap.org/wiki/Recording_GPS_tracks)** to your OSM account. It's a very low effort, completely optional way for users to contribute data to OSM.

So what does a "traditional mapping app" do? These are the things we focused on:

- Show a map
- Find you on that map
- Find things around you
- Get you to those things

Let's get into how we did that.

Show a map

We use our vector tile service to send vector data down to the phone and then **OpenScienceMap (http://www.opensciencemap.org/)** to render it.

Find you on the map

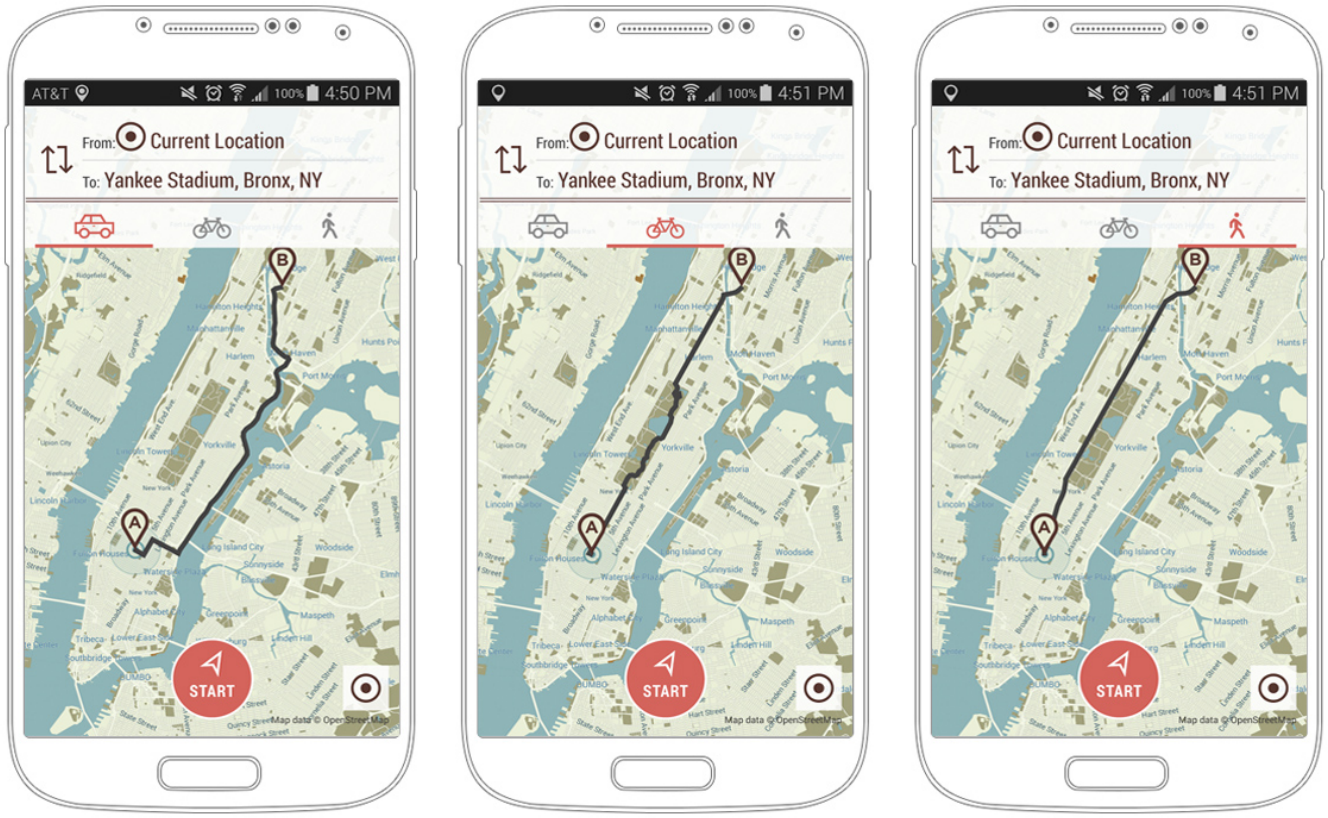
Normally this is a solved problem, but when your mission is to be 100% open source, this becomes slightly trickier. Google does a great job at easily interfacing with Android's location client through Google Play Services. However, Google Play Services is not open source. We wrote a drop in replacement for it called **LOST (https://github.com/mapzen/lost)** which interfaces directly with Android's location client.

Find things around you

For search and geocoding we use **Pelias. (https://github.com/pelias)** Pelias is a modular open source geocoder that uses ElasticSearch for fast autocomplete and forward and reverse geocoding. It's designed to support a variety of datasets, including Geonames, Quattroshapes, OSM, and whatever else you can throw at it.

Get you to those things

Once you have something on a map, the next step is getting there. Enter **Open Source Routing Machine (http://project-osrm.org/)**, which does exactly what you expect. Built and maintained by Dennis Luxen, it does driving, walking and biking directions over the OpenStreetMap road network. It allows us to provide the turn-by-turn directions which users have come to expect. We provide our own **instance of OSRM (https://mapzen.com/blog/osrm-services)**, which also powers Open.



Open is a proof of concept, meant to show the state of open source mobile mapping tools and open data. In some cases, it demonstrates how far open source geo has come. In others, it demonstrates that open source still has a way to go before achieving parity with proprietary services. When an app is built on open data, that data is only as good as the community supporting it. And the great thing about open source is that you can actually do something about it. Open is a starting point, and as more users (users like *you*, maybe) contribute to the libraries, tools, and data source it uses, it can only improve. We built Open so its components could be improved upon by a larger community, and we're looking forward to seeing what you create.

· 17 November 2014 ·

Mike Cunningham

I'm Street View famous in two cities and I do product @mapzen.

On The Road: an Android location snapper

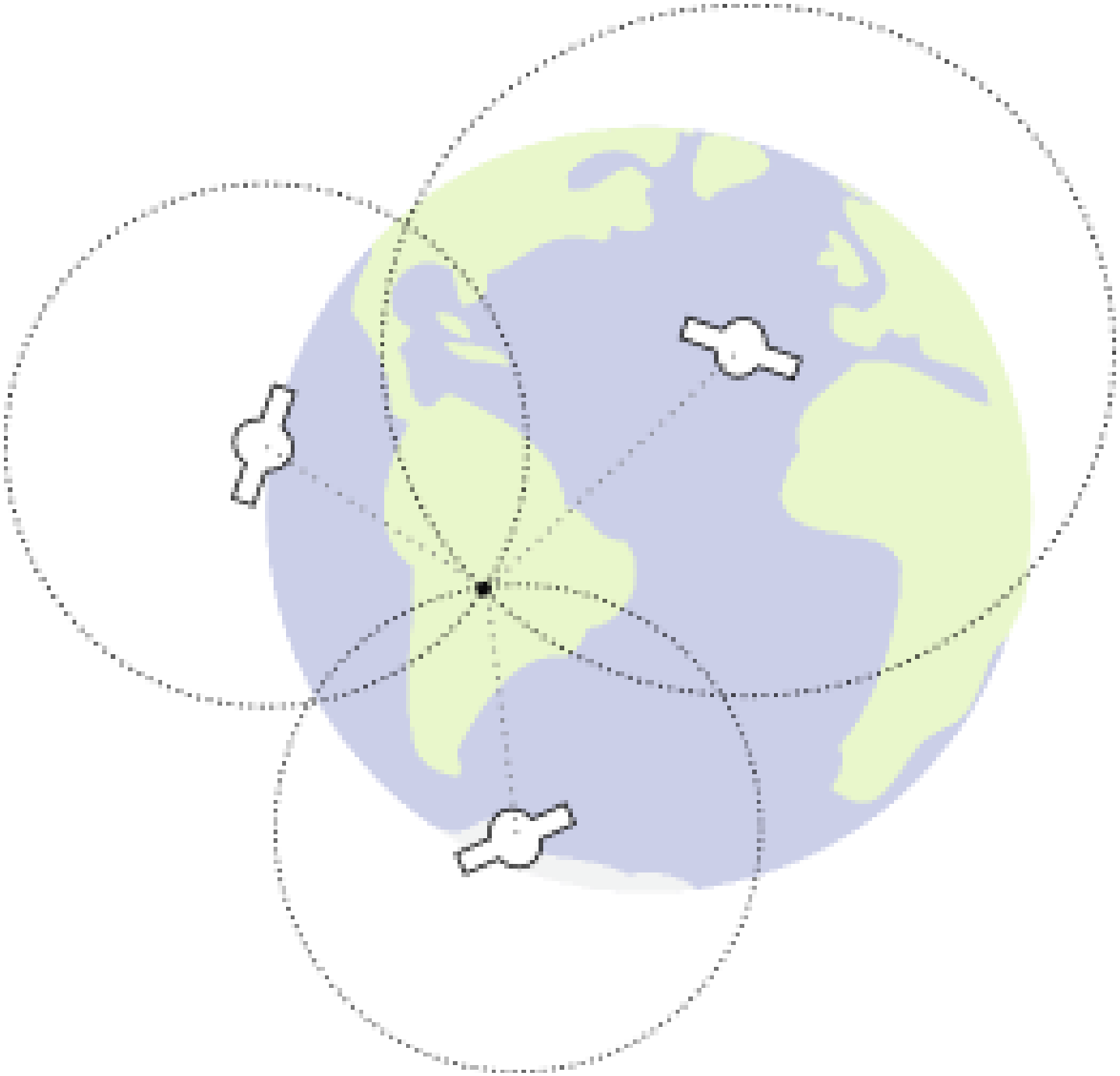
mobile (/tag/mobile) android (/tag/android)

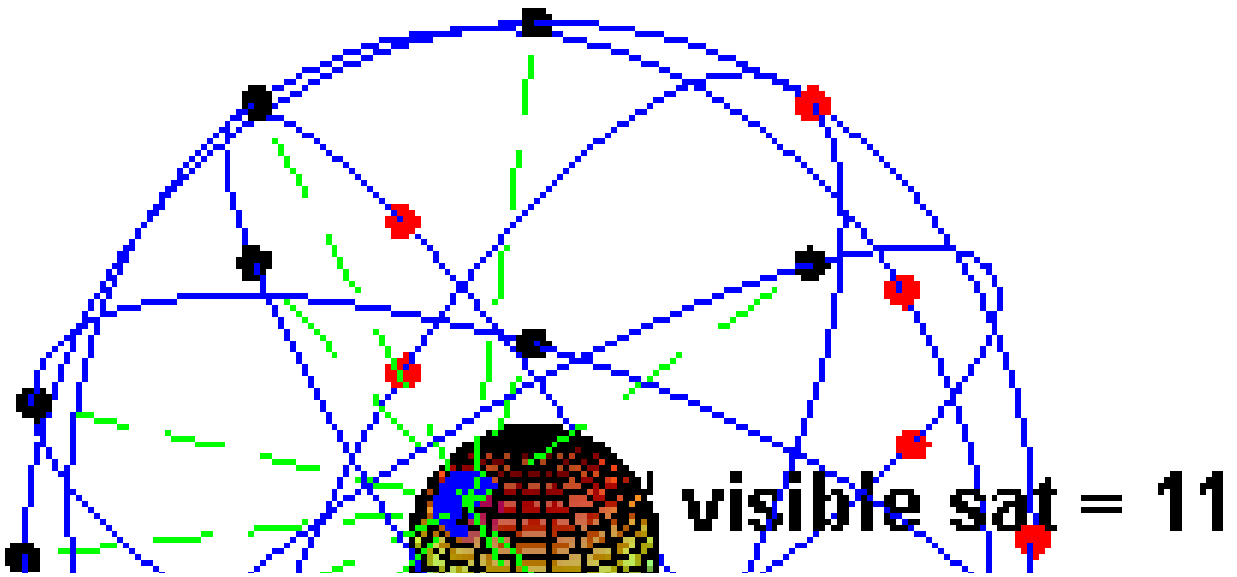
Recently, we released an Android app called **Open (https://play.google.com/store/apps/details?id=com.mapzen.open)**, a location application built with **open data using open source code (https://github.com/mapzen/open)**. We made a number of library components for Open, which we will write about in the coming weeks. Today, we're starting with a library called **On The Road (https://github.com/mapzen/on-the-road)**.

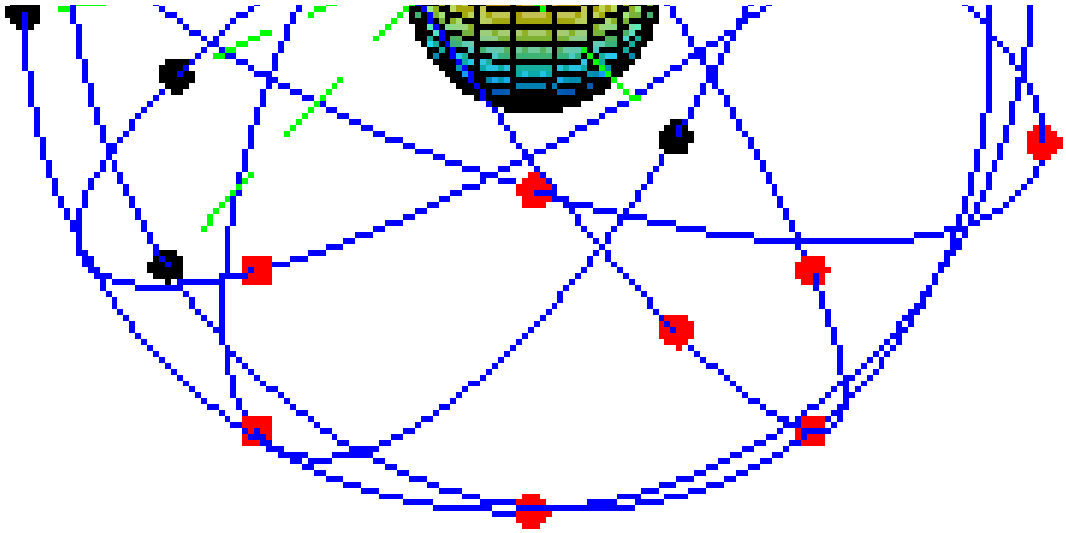
The purpose of this library is twofold: it's a client for **The Open Source Routing Machine (OSRM) (http://project-osrm.org/)** and it handles some of the client-side challenges of routing, such as location snapping (map matching), distance calculations, and turn prompting. This post will cover location snapping which is why we started this library, then cover some of the library usages and some utilities in development to make it easy to develop it further (hopefully, with your contributions).

Location correction

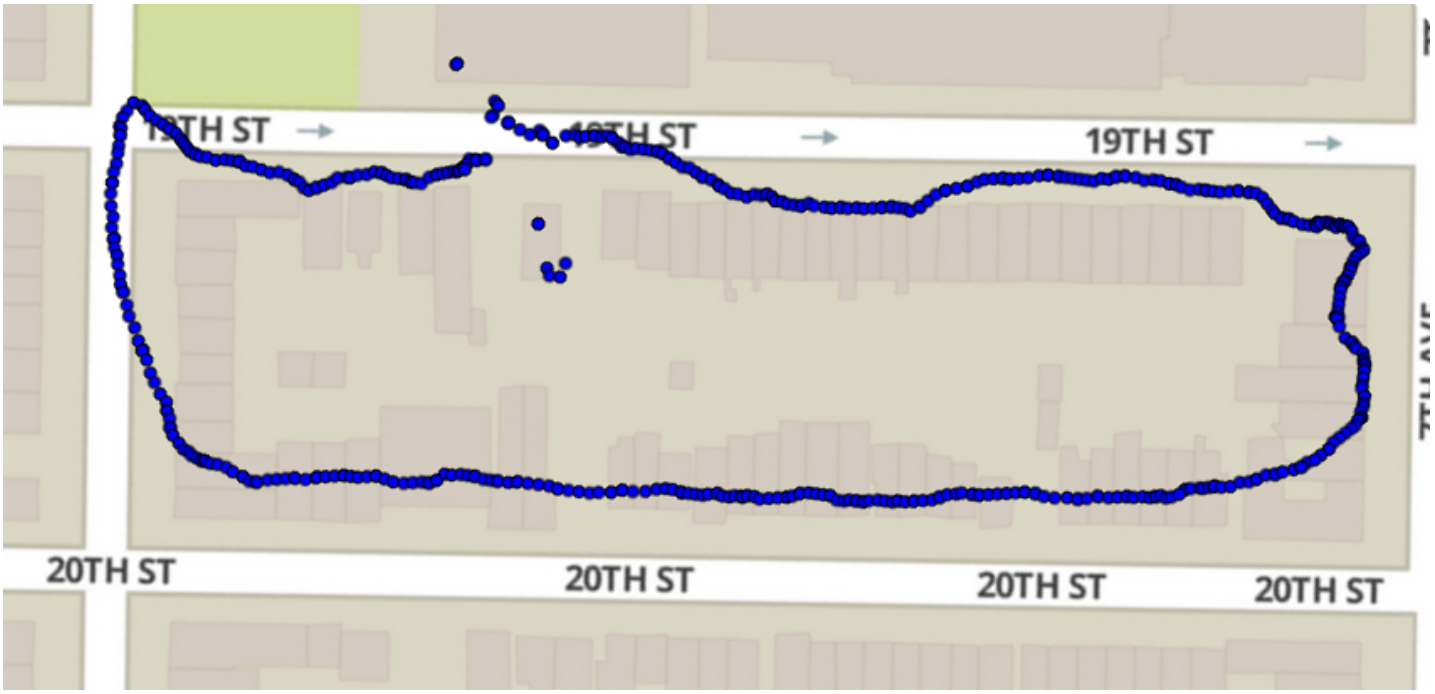
GPS is an amazing technology, and it's important to understand how it works and its limitations. Devices like smartphones have a chip that receives signals from GPS satellites, which transmit their position in space to the chip. When the GPS chip has this information from at least 3 satellites, it triangulates its own position on the planet. Kind of like this:




visible sat = 11



Although technically only 3 satellites are needed for positioning, the more satellites, the more precise the location. Repeated signals from the same set of satellites also have positive effects. If a GPS receiver is on the move and has to switch satellites, or only sees a limited number of them, the quality will suffer. This is very common in cities, where buildings will obstruct direct lines of sight to satellites, limiting the number of satellites to receive signals from and causing frequent switching. Here is a brief example of me walking around my block in Brooklyn.



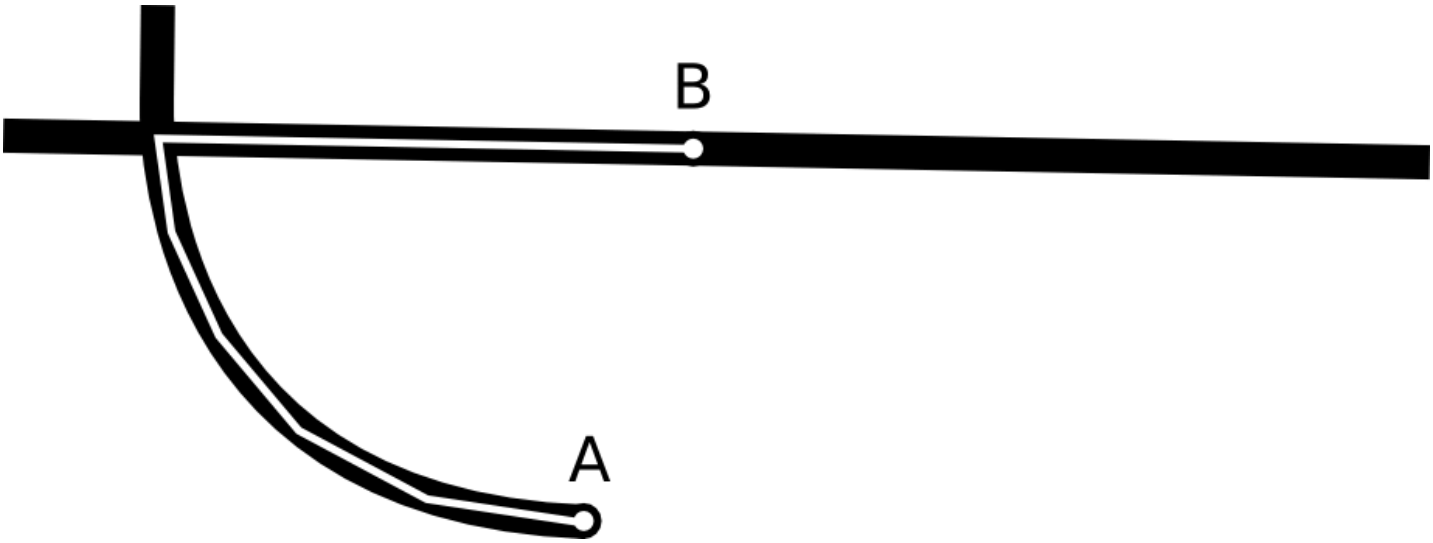
The results are even worse in Manhattan, where buildings are typically higher.

There are a couple of other things to consider when interpreting GPS. First, the accuracy has little to do with your location on the planet and more with how reliable subsequent readings are. On a map, the accuracy reading coming back from the Android location is represented as a circle

with a radius of x meters, within which there is a 68% probability that the true location is contained.

During routing, for the location indicator to appear to be actually on the road it becomes necessary to correct GPS locations. The technique to accomplish this has been called map matching and is an active field of study in which various algorithms have been developed. This library takes a shortcut, since we have a polyline, the route, that we expect the user to follow. The more sophisticated algorithms will snap location onto a road network without a route, which is accomplished by taking various heuristics into account such as speed and direction.

Here is a simplified animation of how the location snapping works.



We maintain a reference to which segment of the polyline we are working with. We take the original location (Red dot) and find where it would intersect with the current segment (dashed line), this is done by taking 90 degrees off the azimuth (bearing) of the original segment. Once we have that we snap location to the route (Green dot). To see if we have traveled past the segment we measure the distance from the beginning of the segment and if it's further than the segment distance we consider it to be on the next segment and do the same calculation to set the fixed location. The math formulas we use were translated from **Chris Veness scripts (http://www.movable-type.co.uk/scripts/latlong.html)**. His examples are written in javascript and we translated them to java to use on Android.

How to use

Adding this to your app is simple, and documented in the **repo (https://github.com/mapzen/on-the-road/blob/master/README.md#install)**. You can download the aar, and you have the option to include this in your maven or gradle build process, as On The Road is hosted on maven central.

By default the library is configured to use our **OSRM instance (/blog/osrm-services)** but this can be changed. The primary class your code will interact with is the Router. You can get a singleton instance by calling Router.getRouter(). With that instance you can daisy chain options to it. For instance, to change the api endpoint call setEndpoint(String).

```
Router.getRouter().setEndpoint("http://example.com")
```

With the router instance, options can be set for the next route.

```
router.setDriving() // Driving is default (setBiking, setWalking also available)
```

A minimum of two locations are required to make up a route and the order of which they are passed in matters.

```
router.setLocation(new double[] {lat, lng})
    .setLocation(new double[] {lat, lng})
    .setLocation(new double[] {lat, lng});
```

Zoomlevel is used to determine the fidelity of the route polyline.

```
router.setZoomLevel(17) // defaults to 17
```

When all options have been set a callback is attached to be used when the request to OSRM is completed.

```
router.setCallback(new Callback() {
    @Override
    public void success(Route route) {
        // do stuff
    }

    @Override
    public void failure(int statusCode) {
        // do stuff
    }
});
```

Finally, when it's time to go execute fetch on the router and request will be made to the OSRM service.

```
router.fetch();
```

The router is a singleton which means that when you want to execute another route you'll need to clearLocations.

```
router.clearLocations();
```

Assuming the request is successful and there are no errors to handle, the success method will be called with an instance of the Route class.

The route instance decodes the encoded polyline OSRM embeds the response and the getGeometry returns a list of node objects which is container for information for each vertex and the segment following it. These nodes can be used to draw the route on a map.

```
List<Node> nodes = route.getGeometry();
```

To get a corrected location based on the original location call:

```
Location onTheRoad = route.snapToRoute(originalLocation);
```

And to get a list of instructions:

```
List<Instructions> instructions = route.getRouteInstructions();
```

There are number of other methods pertaining to distances and to which instruction the user is currently on:

```
route.getDistanceToNextInstruction();
route.getCurrentInstruction();
route.getRemainingDistanceToDestination()
```

These will give you real time information as location changes through snapToRoute method.

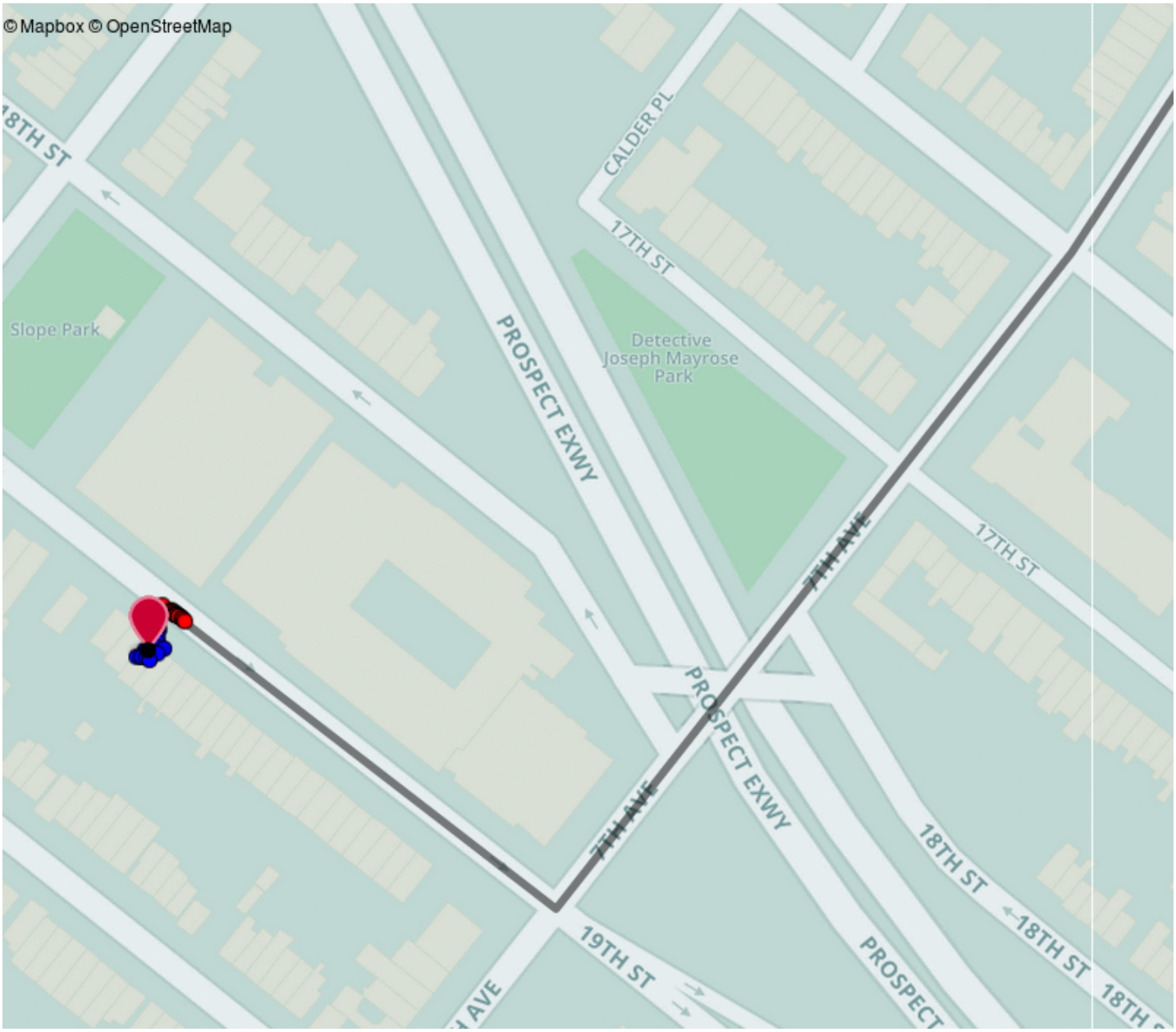
The instructions method has useful properties as well:

```
Instruction instruction = route.getRouteInstructions().get(0);
instruction.getTurnInstruction();
instruction.getHumanTurnInstruction();
instruction.getDistance();
instruction.getDirection();
instruction.getBearing();

//formatted instruction
instruction.getFullInstruction();
```

How to develop / contribute

Testing locations has historically been a painful process, but we have a couple of ways we have been exploring to improve the experience. Beyond unit testing, which is the easiest, we made our own location client that is in a library called **Lost (https://github.com/mapzen/lost)**. We'll be writing more about Lost in the coming weeks, but suffice it to say this library has lots of benefits for both usage and testing. For testing, you can essentially load up a gpx trace, replay locations, and then verify if your code is doing what it's supposed to do. Another utility that we are working on is a **web tester (https://github.com/mapzen/on-the-road_web-tester)** which one would run as developing to test. This is a work in progress, but here is an animation of how it works.



We welcome contributions and would love to have a dialogue about how this library can be generalized further beyond just what we needed for the Open app. Get in touch if you use any of the libraries, and help us make them better!

· 24 November 2014 ·

Baldur Gudbjornsson

Former VP of Engineering, with special focus on devops, infrastructure, community and api management.

Experiments in Crowdsourcing Point Clouds For 3D Maps

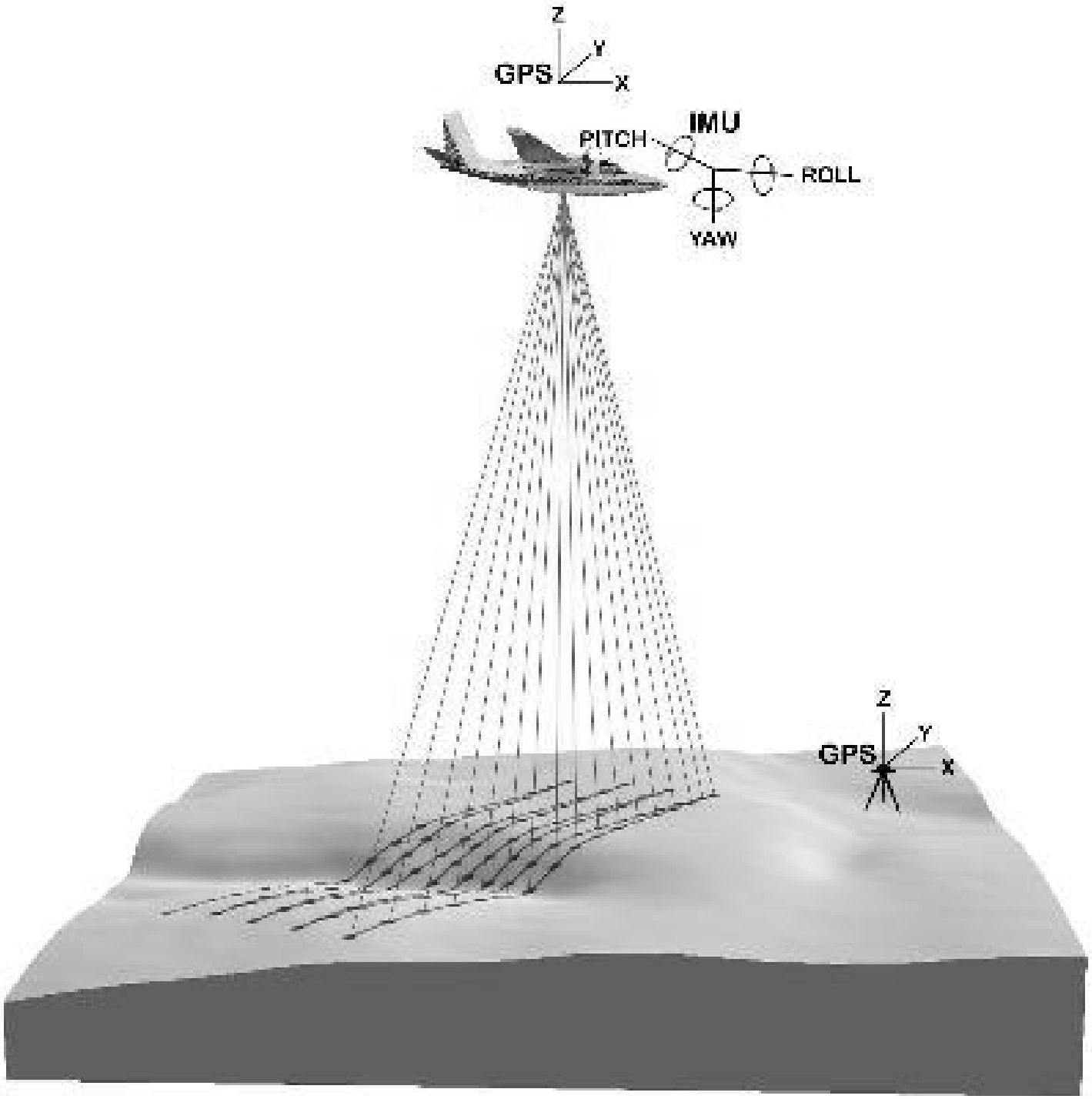
tutorial (/tag/tutorial)

"...the only place on earth where all places are—seen from every angle, each standing clear, without any confusion or blending" "El Aleph", Jorge Luis Borges

What if people could contribute to open source 3D maps just by taking and sharing photos? These experiments with generating point clouds are an attempt to figure out how to make that possible.

Point Clouds and LIDAR

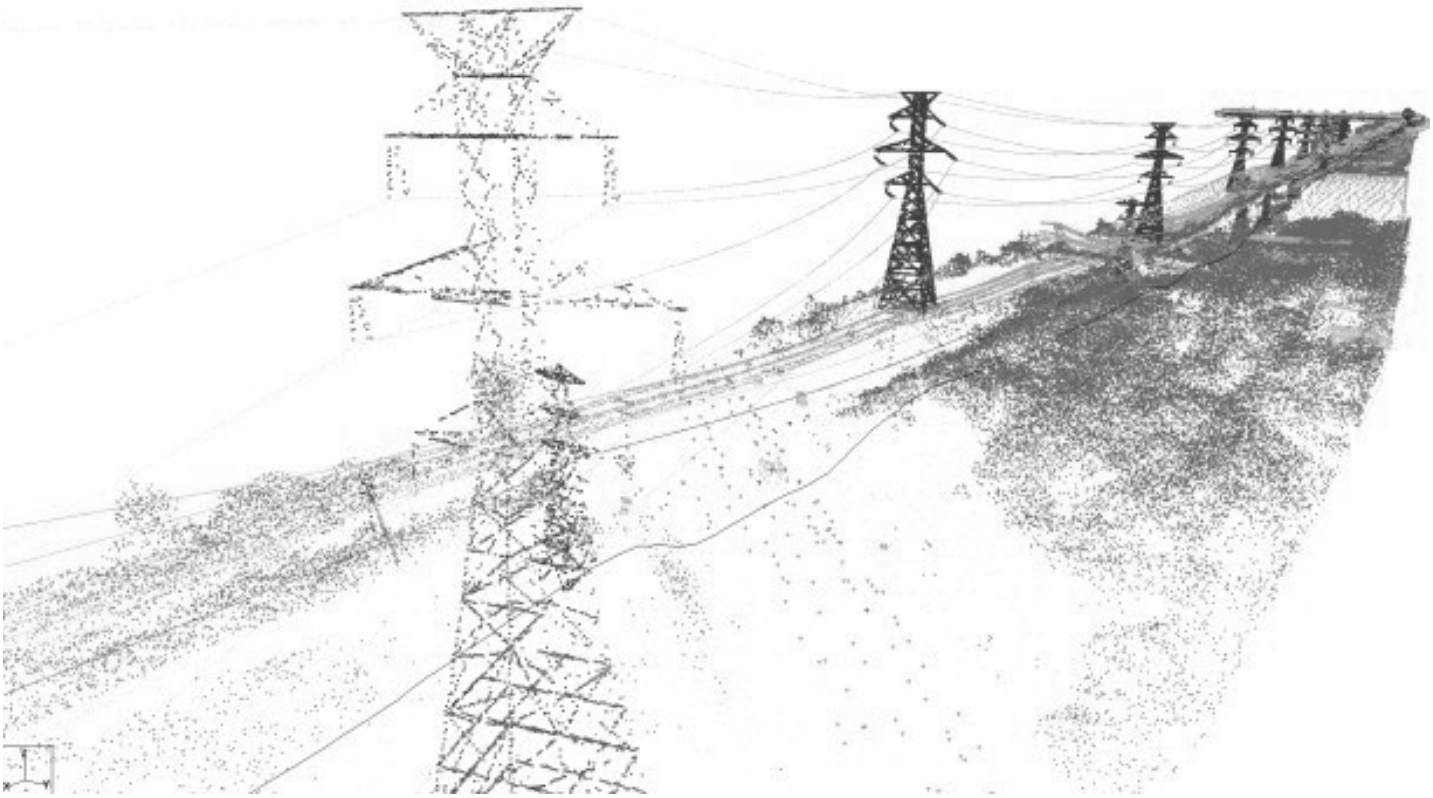
Point clouds are not new, but new devices with more computational power and memory have recently made point cloud tech more accessible. From video games to **the big screen (https://www.youtube.com/watch?v=p6NNQ3VAb3w&list=UUjnYk44Aj9E634TPucpIXnQ)**, more and more points are being generated. Similarly, LIDAR technology isn't all that new—for almost as long as laser technology has existed, scientists have been mounting laser scanners under planes to sends pulses of light to a surface and measure the precise distance to it. LIDAR is the technology that geolocates those points measured by those lasers. (It also seems to be interchangeably written as LIDAR, LiDAR, and lidar; I'm just going to use LIDAR because I LIKE SHOUTING.)



In the last few years, a new generation of smaller LIDAR devices have come to the market. They can be mounted on cars and backpacks, preserving every single point from a particular place—like holographic glass snow globes.

According to the closing law of gestalt psychology, human perception is very good at making sense of these clusters of dots. Our mind completes the empty spots. The more perspectives the sources have, the closer the point cloud map looks to the terrain.

In order to work with LIDAR data, you first have to merge and clean it. There are a couple of programs that help you do this at this scale; the best open source application I found was **CloudCompare (http://www.danielgm.net/cc/)**.



Once you've cleaned your LIDAR data, you have to think about serving and analyzing the data. **PostGIS (http://postgis.net/)** can be used to combine this LIDAR information with any map. A great introduction to this process is **this tutorial from Yuriy Czoli (https://gist.github.com/YKCzoli/3605e014b8ed09a571e5)** where he uses the altitude of points to set the height of building on **OpenStreetMap (http://www.openstreetmap.org/)**.

In my case, I was interested in creating ways for **OpenStreetMap (http://www.openstreetmap.org/)** polygons to look more accurate. First I had to create **some scripts (https://github.com/tangrams/LIDAR-tools)** to convert LIDAR data to other formats, then **export them to PostGIS (https://gist.github.com/patriciogonzalezvivo/229c5cd4001c2ed45ec6)** and extract single buildings or tiles from them.

All of these tools you can find at this **LIDAR-Tools repository (https://github.com/tangrams/LIDAR-tools)**:

- `las2SM` : re-project LIDAR files (.laz/.las) to **Spherical Mercator (epsg:3857) (http://epsg.io/3857)**

- `las2tile` : crop a tile from a LIDAR file (.laz/.las) and project to **Spherical Mercator (epsg:3857) (http://epsg.io/3857)**
- `las2ply` : export LIDAR files (.laz/.las) into .ply format
- `getPointsForID` : once a PostGIS database is loaded with LIDAR information of a region, this script gets all the points inside a tagged OSM polygon by providing the OSM ID.

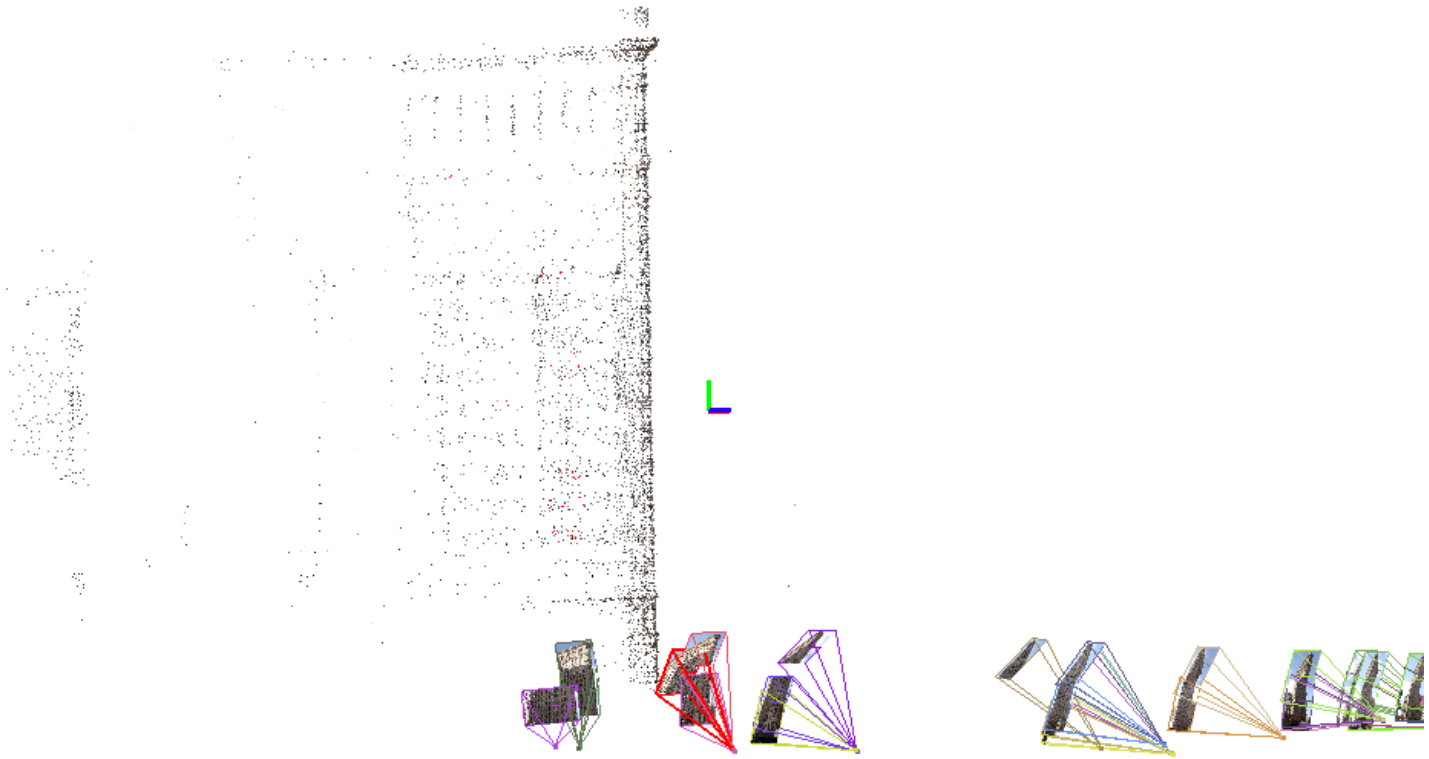
Making point-clouds from pictures

But how many people just happen to own a LIDAR? They are expensive devices that only companies and governments can afford. As far as I know, that's how "open" this technology is.

There is a way to achieve similar point cloud results without LIDAR, using a relatively older technique called photogrammetry, which requires just a regular camera and some patience (especially to compile the available open software). It stitches photographs together, finding distinctive assets and cross-similarities to construct 3D point clouds. Today, the most popular applications using this technology are **PhotoSynth (https://photosynth.net/)** and **123D Catch (http://www.123dapp.com/catch)**. The are also some powerful open source equivalents, such us **Bundler (http://www.cs.cornell.edu/%7Esnavely/bundler/)** and **VisualSfM (http://ccwu.me/vsfm/)**.

My experiments with photogrammetry began with fieldwork: taking pictures of the Flatiron building. I followed **Jesse Spielman's tutorial (http://wedidstuff.heavyimage.com/index.php/2013/07/12/open-source-photogrammetry-workflow/)** to learn how to take better pictures, process them with **VisualSfM (http://ccwu.me/vsfm/)**, and clean the dense reconstruction with **MeshLab (http://meshlab.sourceforge.net/)**.

I found this to be a really smooth and flexible work flow. Pictures can be taken in different moments of the day and with any camera. In my case, I finished adding a bunch of them with my cellphone because I was interested in using its GPS geolocation (which ended up being a whole new problem).

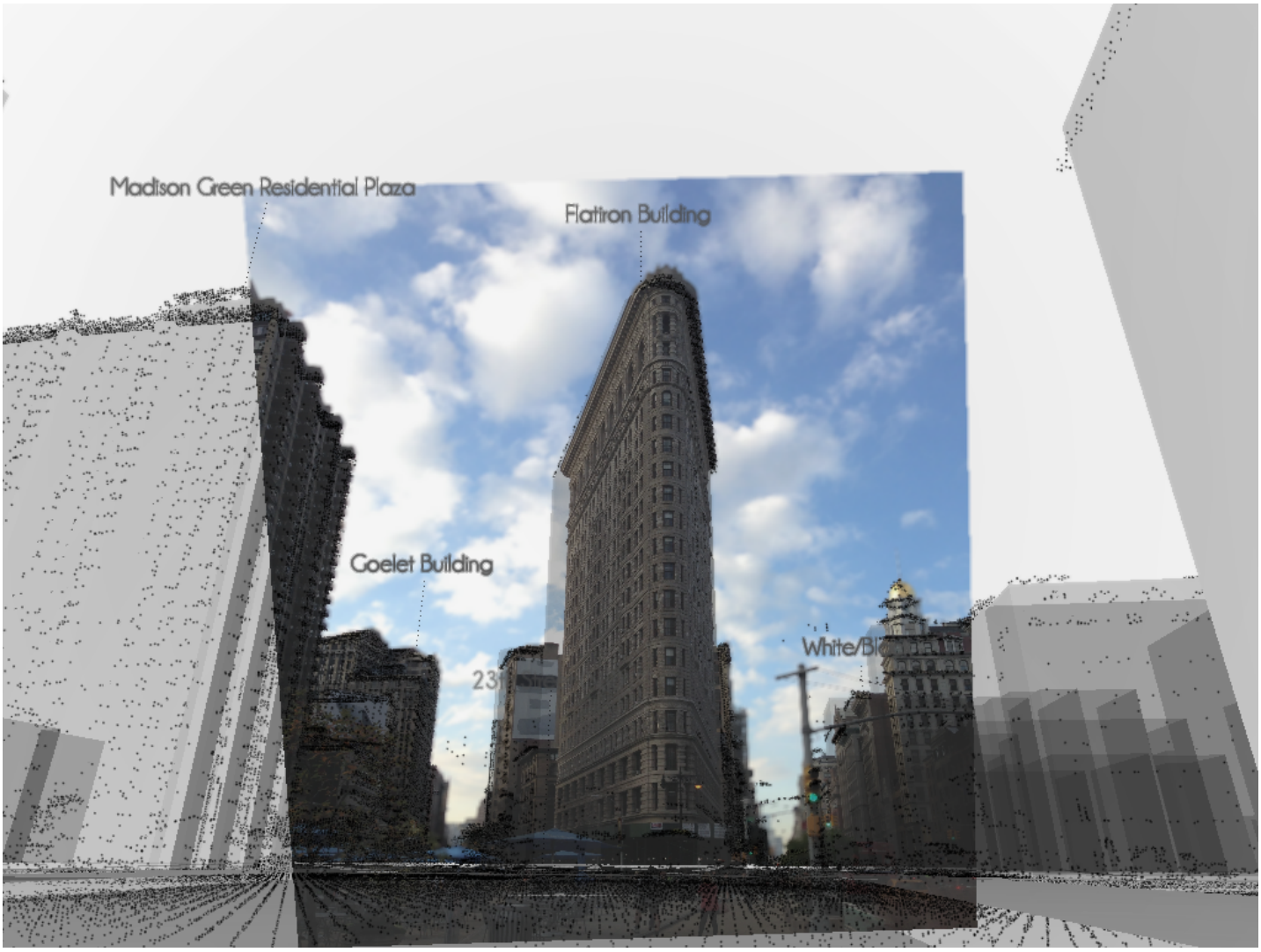


The results of the above process are: a point cloud of feature markers that contain information from the cameras called `bundle.rd.out` (from the **bundler (http://www.cs.cornell.edu/%7Esnavely/bundler/)** algorithm) and a big point cloud product of the dense reconstruction (from the **pmvs2 (http://grail.cs.washington.edu/software/pmvs/)**). Both files are coherent between each other. They share the same space coordinates. You can load them and experiment with them using **this addon (https://github.com/patriciogonzalezvivo/ofxBundle)** for **openFrameworks (http://openframeworks.cc/)**.

In order to import this data constructed just from photos to PostGIS, I had to find a way to georeference the point clouds. After taking more photos with my cellphone and using the GPS location hidden on the **EXIF header (http://www.digicamhelp.com/glossary/exif-data/)**, I was able to extract the centroid, approximated scale, and base rotation to level the cameras over the surface. The final adjustments (translation and precise orientation) were made by hand because

of the incredible noise in GPS data thanks to the well-known **“urban canyon” (http://en.wikipedia.org/wiki/Street_canyon)** effect that makes satellite signals bounce between buildings.

The following images show the result of this process. The Flatiron points, photo images and extruded OSM polygons are displayed together. The abstract universe of maps is contrasted side by side with photographs and virtual points.

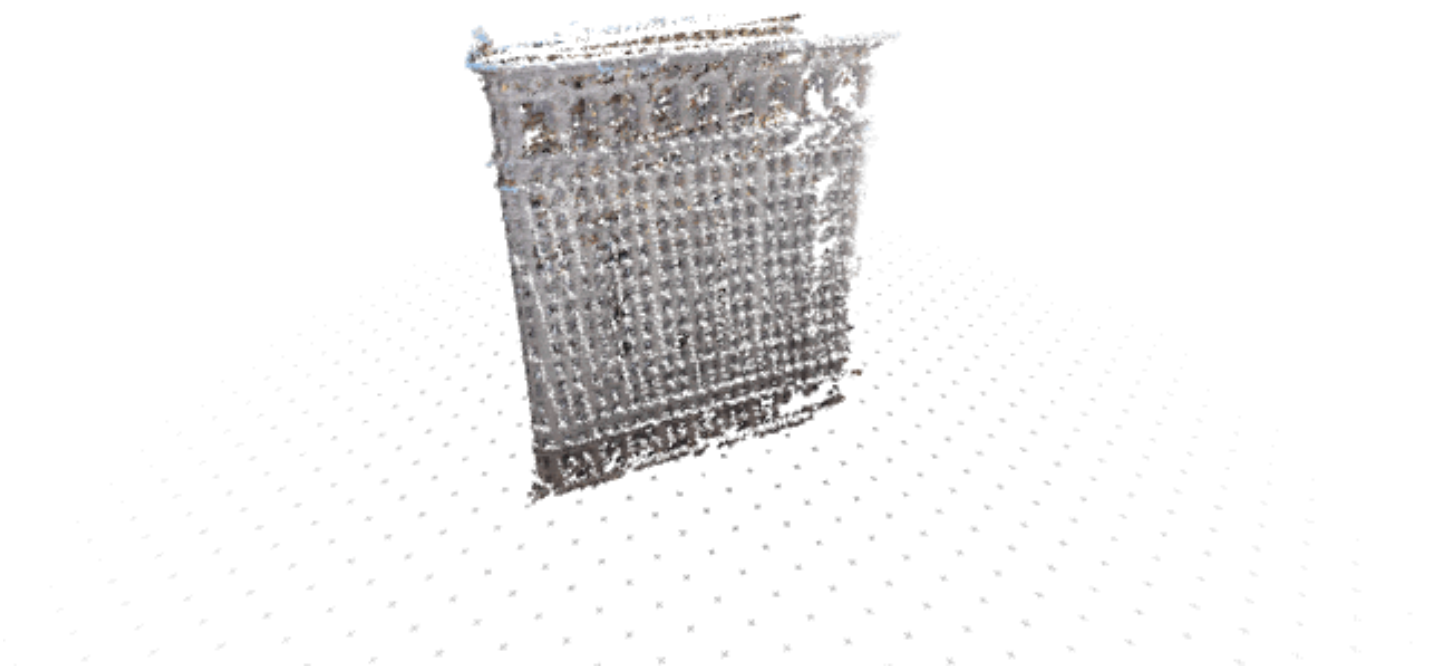


From Point Clouds to Meshes

Once I got a coherent universe of point clouds correctly geolocated, it was time to load them into my PostGIS database together with an **OSM Metro Extract (https://mapzen.com/metro-extracts)**. To do that, you can follow **this short tutorial (https://gist.github.com/patriciogonzalezvivo/229c5cd4001c2ed45ec6)**. The reason to load the Metro Extract? You can use PostGIS geo-spatial query functions to extract with surgical precision a specific feature, such as the "Flatiron Building" or `node:2517056822`. Inside **the repository LIDAR-Tools (https://github.com/tangrams/LIDAR-tools)** you will find a Python script call `getPointsForID` that will let you do this extraction without knowing how to make a PostGIS query.

With all the points of the Flatiron I was able to perform a Poisson Reconstructive Algorithm (using **CGAL's implementation (http://doc.cgal.org/latest/Surface_reconstruction_points_3/)**) to construct a mesh from the points. You can find this tool at **the LIDAR-Tools repository (https://github.com/tangrams/LIDAR-tools/)** in the sub folder called **`xyz2Mesh` (https://github.com/tangrams/LIDAR-tools/tree/master/xyz2mesh)**. Alternatively you can use **MeshLab's (http://meshlab.sourceforge.net/)**, but I found that CGAL's implementation works better.

PointCloud

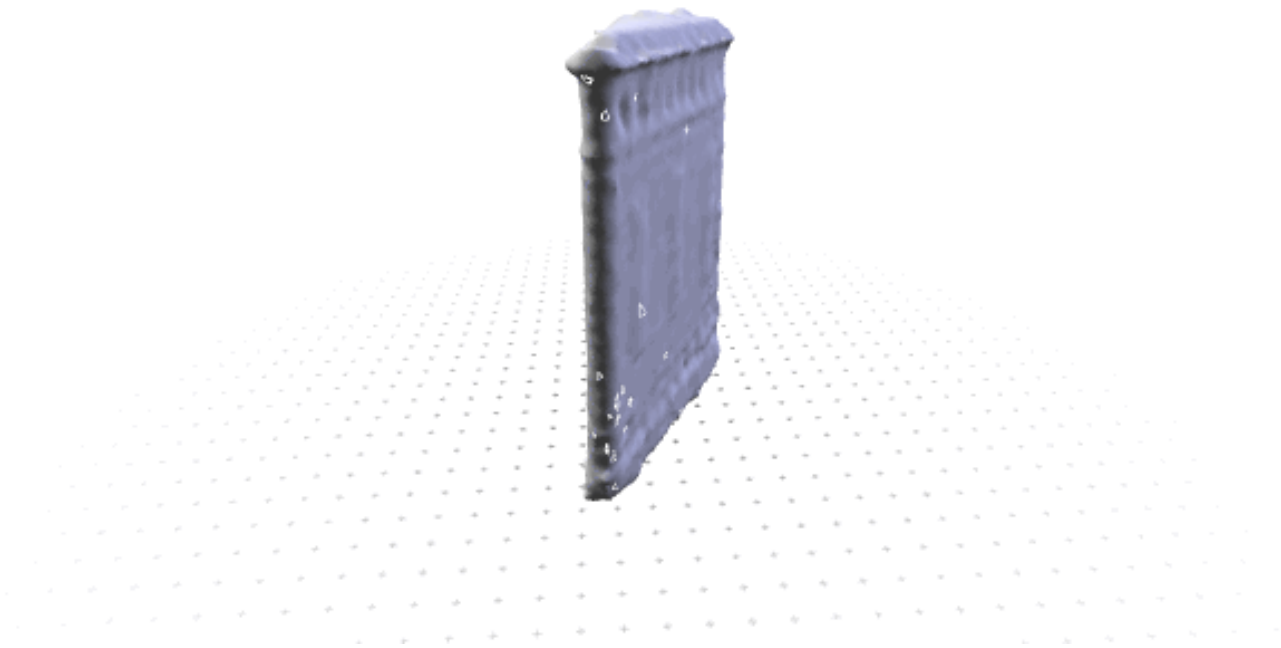


Meshes to GeoJSON Polygons

Although I love how meshes look, they have too much information for what OSM can hold. This last step in the process is about downsampling the information we collect to something that can be uploaded and shared on the web.

OSM's database isn't really designed for 3D meshes. Basically all buildings are polygons with altitudes. Like layers of a cake, the polygons sit one on top of another. In order to make our mesh compatible with OSM we have to cut it into "slices" like a layer cake. In **this repository (https://github.com/tangrams/Mesh2OSMSlicer)** you will find a program that does exactly that. It slices the mesh inch by inch, checking every slice with the previous one, if it finds a significant change on the area size, it adds that cut on top of our cake.

Mesh



Next Steps

For this experiment I used an extraordinarily convenient sample. Beside being a beautiful building, the Flatiron is perfectly isolated construction. To apply this process to other buildings, extra steps have to be added to the pipeline. This was a proof of concept investigating the potential of mixing LIDAR information with SfM reconstructions as a way to crowdsource point clouds. While there's still a lot of work to do to make that process simple and easy for the average user, these results and their potential applications are exciting.

· 01 December 2014 ·

Patricio Gonzalez Vivo

Patricio is an artist and graphic engineer who speaks the language of light.